Spaß mit Computern

MASCHINEN CODE

Für Z 80 und 6502

Ein Buch von CHIP

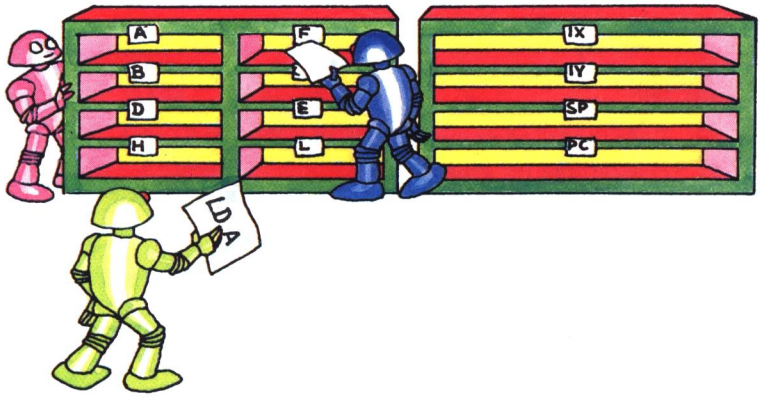
Ravensburger®

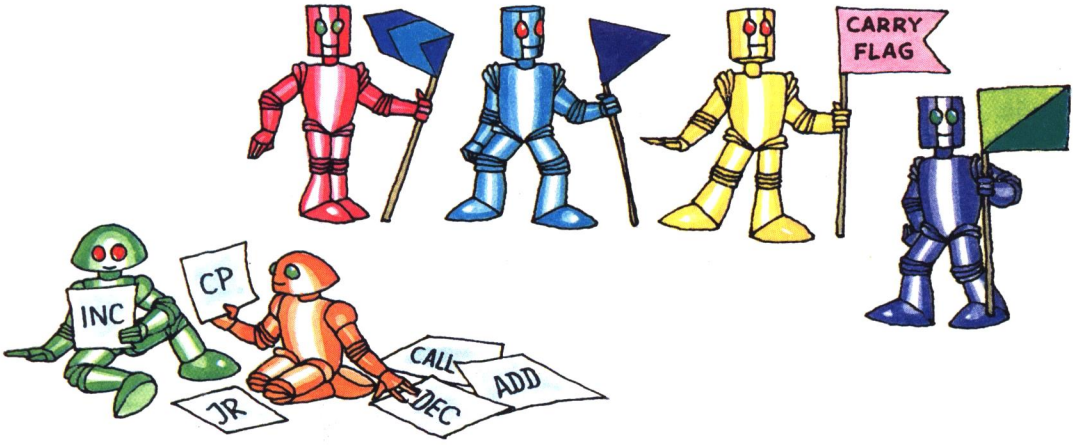
Lisa Watts · Mike Wharton

MASCHINEN CODE

Für Z80 und 6502

Aus dem Englischen übersetzt von
Cornelius Siegel





Otto Maier Verlag Ravensburg
Vogel-Verlag Würzburg

Inhalt

3 Über dieses Buch
4 Was ist Maschinencode?
6 Was im Computer vorgeht
8 Der Speicher des Computers
11 Hexadezimalzahlen
12 PEEK und POKE
14 Blick in die Zentraleinheit
16 Anweisungen für die Zentraleinheit
18 Wie man ein Programm in Hex übersetzt
20 Gesucht: freier Speicherplatz
23 Wie man Programme lädt und zum Laufen bringt
27 Wie man Daten im Speicher findet
28 Wie man mit großen Zahlen rechnet
29 Das Übertragskennzeichen
30 Programme für große Zahlen
32 Wie man Text auf den Bildschirm bringt
35 Sprünge und Verzweigungen
38 Programm „Bildschirm-Blinken"
40 Wie man weiterkommt
41 Umwandlungstabellen für Dezimal- und Hex-Zahlen
42 Mnemonics und Hex-Codes für den Z80
45 Mnemonics und Hex-Codes für den 6502
46 Lösungen
47 Register

Titel der Originalausgabe: Usborne Introduction to Machine Code for Beginners
Aus dem Englischen übersetzt von Cornelius Siegel
Elektronik-Fachberatung: Chris Oxlade, A. P. Stephenson und Martin Stübs
Illustrationen: Naomi Reed und Graham Round
Buchgestaltung: Lynne Norman und Graham Round
Umschlaggestaltung: Achim Köppel unter Verwendung des Umschlags der Originalausgabe

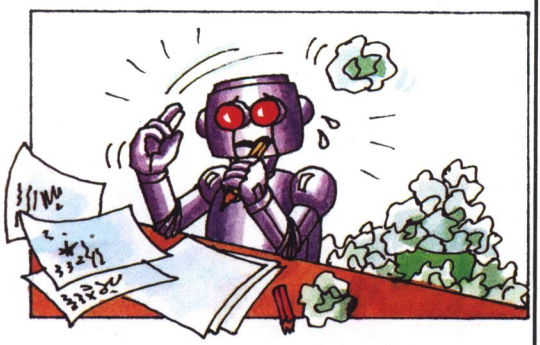
Gemeinschaftsausgabe
Otto Maier Verlag, Ravensburg · Vogel-Verlag, Würzburg

© 1983 by Usborne Publishing Ltd., London
Alle Rechte der deutschen Bearbeitung liegen beim Otto Maier Verlag, Ravensburg, 1985

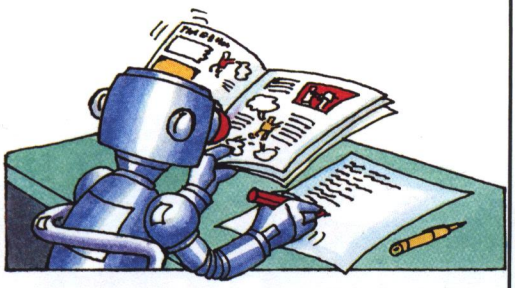
Printed in Spain
ISBN 3-473-35613-1 (Otto Maier Verlag)
ISBN 3-8023-0822-0 (Vogel-Verlag)

Über dieses Buch

Dieses Buch gibt Schritt für Schritt eine verständliche Einführung in das Programmieren in Maschinencode. Maschinencode ist der Code, mit dem der Computer direkt arbeitet. Programme in diesem Code laufen schneller und brauchen weniger Speicherplatz als solche in BASIC. Sie sind allerdings etwas schwieriger zu schreiben und zu verstehen als BASIC-Programme.

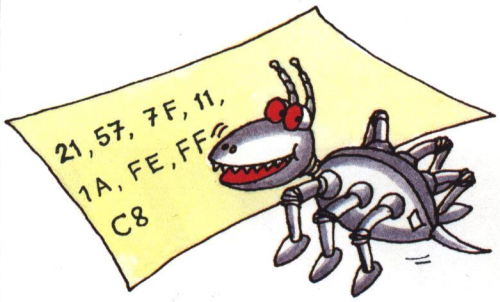


Das Buch erklärt zunächst die Grundprinzipien des Maschinencodes. Es zeigt, wie man kleine Maschinencode-Programme schreibt, wie man sie in den Computer laden und ablaufen lassen kann.

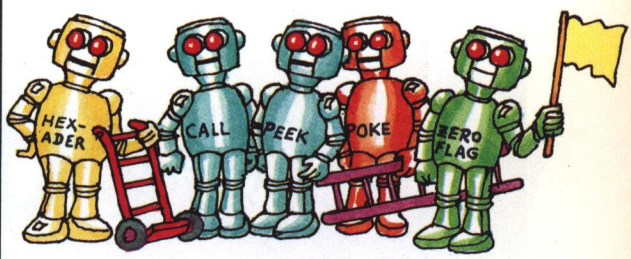


Das Buch wurde speziell für Computer mit den Mikroprozessoren Z80 oder 6502 geschrieben*. (Der Mikroprozessor ist der Chip, der die Zentraleinheit enthält.) Computer mit anderen Mikroprozessoren arbeiten mit unterschiedlichen Maschinencodes; Computer mit dem gleichen Mikroprozessor verwenden jedoch den gleichen Code.

Wenn man in Maschinencode programmiert, muß man viele Regeln beachten und Einzelheiten im Kopf behalten. Das ist anfangs ziemlich anstrengend und manchmal auch verwirrend, weil man Programme in Maschinencode nicht ohne weiteres lesen kann. Sie bestehen nämlich nur aus Zahlen und Buchstaben. Fehler sind schlecht zu finden und können unangenehme Folgen haben. Man muß daher sehr sorgfältig arbeiten.



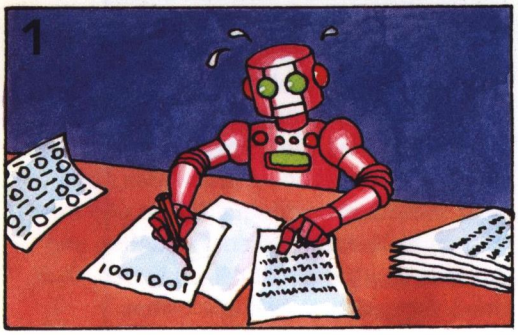
Man sollte schon ein wenig Ausdauer haben, sonst hat es keinen Sinn, längere Programme in Maschinencode zu schreiben. Einige Dinge lassen sich ebensogut in BASIC erledigen. Andere Dinge aber, wie schnelle Spiele oder Bildschirmeffekte, erfordern Maschinencode. Dieses Buch zeigt, wie man BASIC-Programme mit kurzen Unterprogrammen in Maschinencode spannender machen kann.



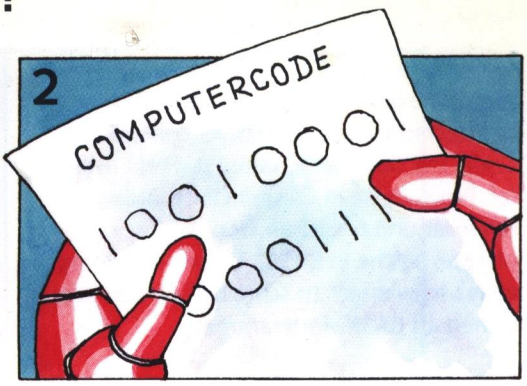
Am Schluß des Buches finden Sie Umwandlungs- und Codetabellen, die Ihnen bei der Übersetzung Ihrer Programme in Maschinencode helfen sollen, sowie die Lösungen für die Rätselfragen und Übungen, die in diesem Buch enthalten sind.

* Die Sinclair-Computer Spectrum und ZX81 enthalten den Mikroprozessor Z80. Der 6502 steckt im VC-20, in Atari-Computern, im Oric und in anderen. Der Commodore 64 hat den 6510, versteht aber den Maschinencode des 6502.

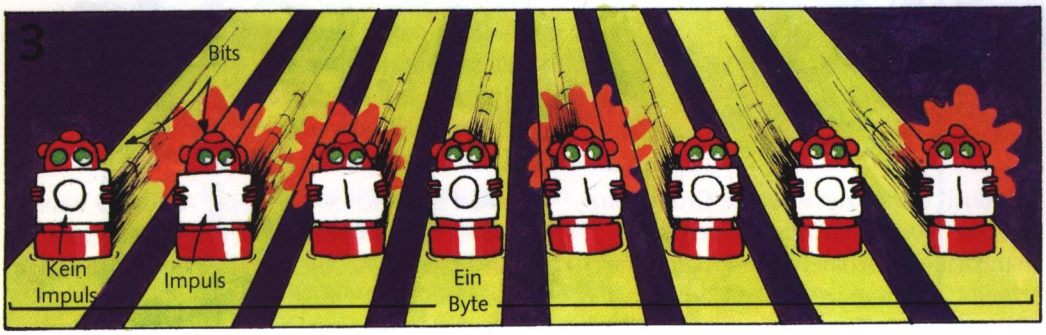
Was ist Maschinencode?



Maschinencode ist der Code, mit dem der Computer arbeitet. Erhält ein Computer ein Programm in BASIC, so übersetzt er alle Anweisungen in Maschinencode, damit er das Programm ausführen kann.

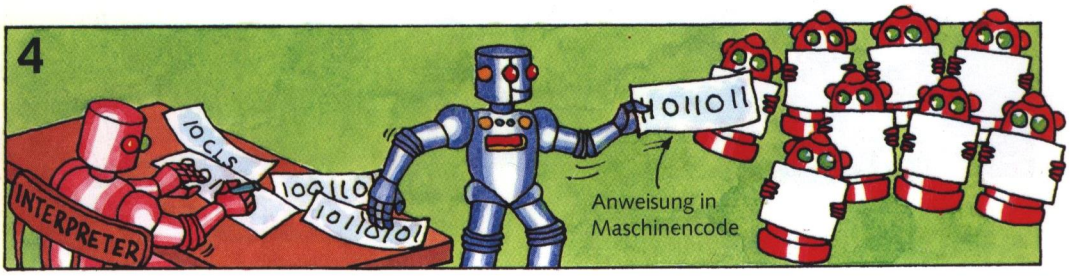


Im Maschinencode wird jede Anweisung und Information als Binärzahl dargestellt. Bei dieser Art der Darstellung werden nur zwei Ziffern verwendet, 0 und 1. Mit diesen beiden Ziffern lassen sich alle Zahlen schreiben. (Mehr über Binärzahlen auf Seite 28.)



Im Computer werden die Binärzahlen aus elektrischen Impulsen gebildet: Ein Impuls bedeutet 1, kein Impuls bedeutet 0. Jeden dieser Impulse oder Nicht-Impulse nennt man ein *Bit* (Abkürzung des englischen Begriffs **b**inary digi**t**).

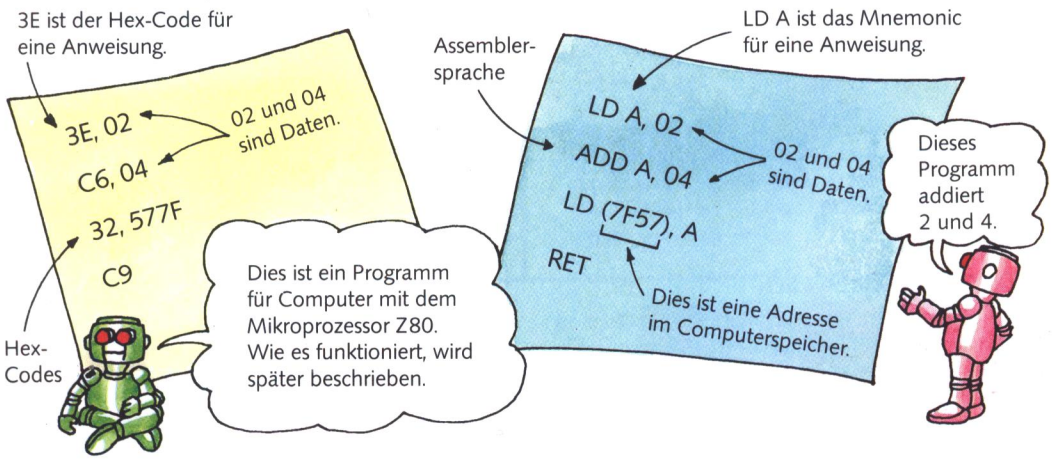
Die Bits bilden im Computer Gruppen von je acht Bits. Eine solche Gruppe heißt *Byte*. Jedes Byte aus Impulsen und Nicht-Impulsen stellt eine Binärzahl für eine Anweisung oder Information in Maschinencode dar.



Jede Aufgabe, die der Computer erledigt, etwa Zahlen zu addieren oder den Bildschirm zu räumen, erfordert eine ganze Folge von Anweisungen in Maschinencode. Gibt man dem Computer eine Anweisung in BASIC, so übersetzt ein spezielles Programm, der *Interpreter*, die Anweisung in den Maschinencode, den der Computer dann „versteht". BASIC-Programme muß der Computer also erst umwandeln, Programme in Maschinencode kann er sofort bearbeiten. Damit er beispielsweise den Bildschirm räumt, muß man ihm alle Anweisungen dafür einzeln geben.

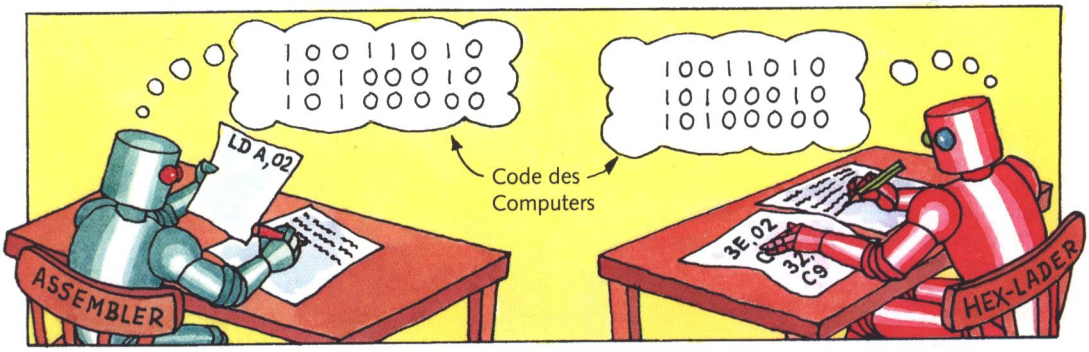
So schreibt man Programme in Maschinencode

Maschinencode-Programme lassen sich auf verschiedene Arten schreiben. Man könnte dazu Binärzahlen verwenden, doch das wäre sehr zeitraubend. Statt dessen benutzt man dazu häufig das Hexadezimalsystem, auch kurz *Hex* genannt. Mit etwas Übung läßt sich das leichter handhaben als Binärzahlen.
Man kann auch in Assemblersprache schreiben. Dann entspricht jeder Anweisung ein Buchstabencode (*Mnemonic*), der an die Bedeutung der jeweiligen Anweisung erinnert.



Dies ist ein kleines Maschinencode-Programm in Hex. Im Hexadezimalsystem werden die Zahlen 0 bis 15 durch die Ziffern 0 bis 9 und die Buchstaben A bis F dargestellt. (Mehr darüber steht weiter hinten im Buch.) Die Hex-Zahl am Anfang jeder Programmzeile bedeutet eine Anweisung. Sie ist der Hex-Code der entsprechenden Binärzahl.

Dies ist das gleiche Programm in Assemblersprache. Jede Zeile enthält das Mnemonic für eine Anweisung, dem die Hex-Zahl in der gleichen Zeile im Programm links entspricht. LD A bedeutet also das gleiche wie 3E, und LD A steht für **load** A, was auf deutsch „speichere (lade) A" bedeutet. Etwas Englisch sollte man also können, damit man die Mnemonics versteht.



Damit man einem Computer ein Programm in Assemblersprache eingeben kann, braucht man ein spezielles Programm, den *Assembler*, der die Mnemonics in den Code des Computers übersetzt. Bei manchen Computern ist der Assembler fest eingebaut, bei anderen kann man ihn auf Kassette kaufen und in den Computer laden. Oder man schreibt ein Maschinencode-Programm in Mnemonics (die man sich leichter merken kann als Zahlen) und übersetzt es selbst in Hex-Zahlen. Manche Computer verstehen Hex-Zahlen direkt. Für andere braucht man einen *Hex-Lader* (siehe Seite 24), der die Hex-Zahlen für den Computer übersetzt.

Was im Computer vorgeht

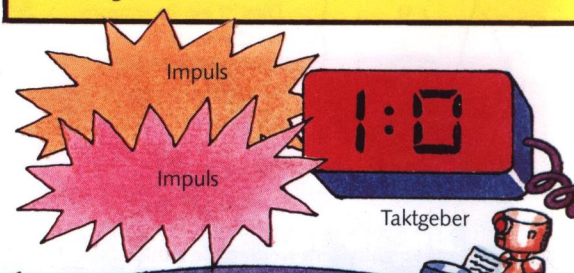
Programmiert man einen Computer in Maschinencode, so muß man ihm jeden einzelnen Schritt genau vorschreiben: wo er Daten findet und speichern soll, was er auf dem Bildschirm anzeigen soll usw. (In BASIC übernehmen das eingebaute Programme.) Dazu muß man ziemlich genau wissen, was im Computer vorgeht. Die Bilder auf diesen beiden Seiten zeigen die Teile im Inneren eines Heimcomputers und wozu sie dienen. Mehr darüber finden Sie auf den nächsten Seiten.

Ein Blick ins Gehäuse

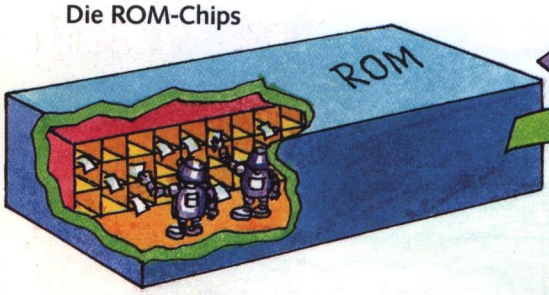
Unter der Tastatur eines Heimcomputers befindet sich eine Platine mit gedruckten Schaltungen; ihre Kupferbahnen leiten den elektrischen Strom. Auf der Platine sind die verschiedenen Chips befestigt.

Chips und ihre Aufgaben

Diese Bilder zeigen, was die verschiedenen Chips im Computer tun. Zwischen den Chips bewegen sich die Informationen als Bytes, d. h. als Achtergruppen von Impulsen und Nicht-Impulsen, die Daten und Anweisungen darstellen.



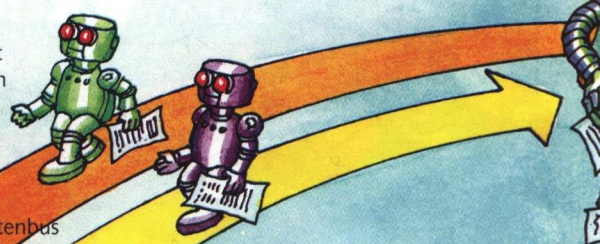
Die ROM-Chips



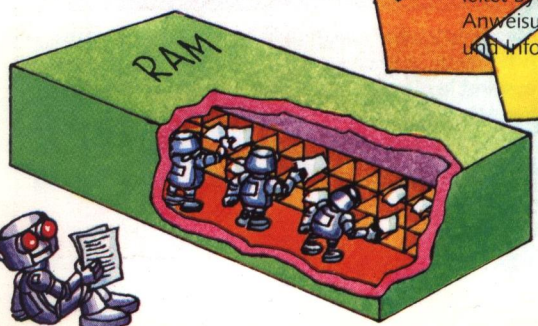
Der Adreßbus leitet Speicheradressen.

Bytes des Computercodes laufen zwischen den Chips auf den Bahnen der Platine. Es gibt drei Gruppen von Bahnen für Impulsgruppen mit unterschiedlicher Bedeutung. Die Gruppen von Bahnen heißen Busse.

ROM ist die englische Abkürzung für **R**ead **O**nly **M**emory und bedeutet etwa „Nur-Lese-Speicher". Dieser Speicher heißt so, weil der Computer die dort gespeicherten Informationen nur lesen kann; er kann dort nichts Neues speichern. In den meisten Heimcomputern enthält das ROM den BASIC-Interpreter, d. h. das Programm, das BASIC in Maschinencode übersetzt.

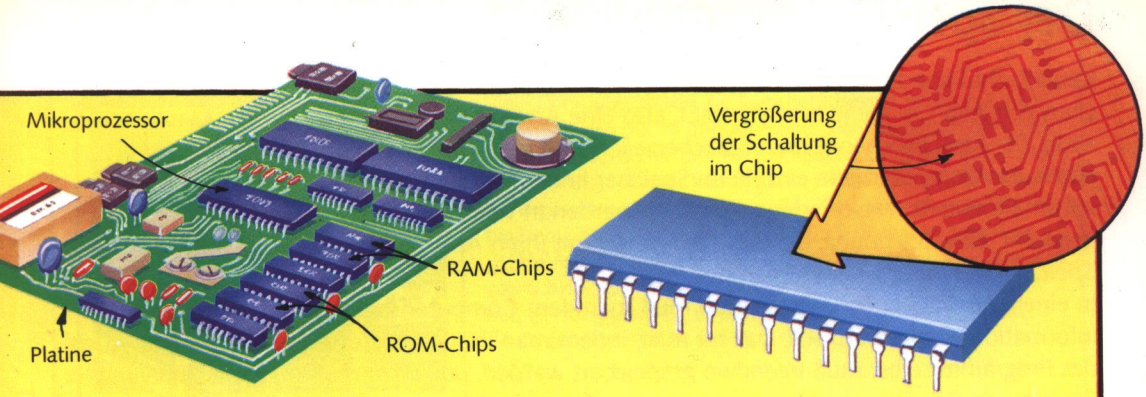


Der Datenbus leitet Bytes für Anweisungen und Informationen.



Die RAM-Chips

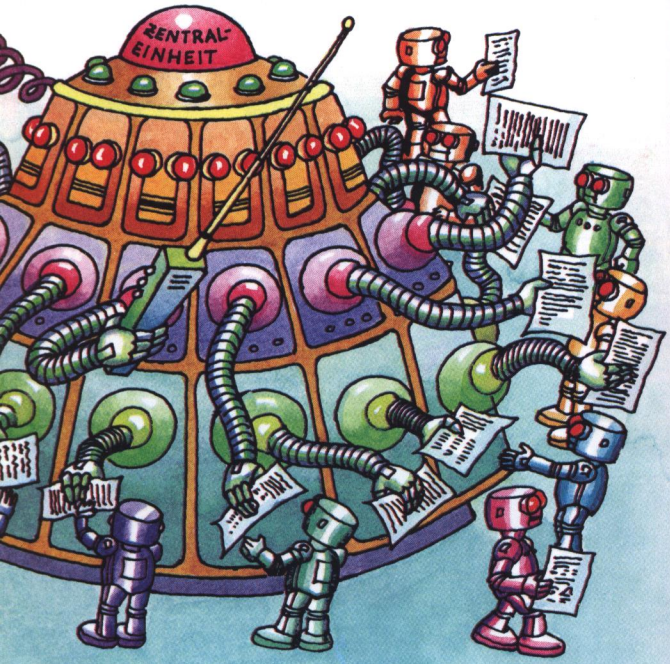
RAM ist die englische Abkürzung für **R**andom **A**ccess **M**emory und bedeutet etwa „Direktzugriffspeicher". Hier werden die eingegebenen Programme gespeichert, mit denen der Computer arbeiten soll, und der Computer speichert dort auch Zwischenergebnisse. Die Informationen in diesem Speicher gehen verloren, wenn man den Computer ausschaltet.



Die genaue Bezeichnung für einen Chip ist *Integrierte Schaltung* (oft abgekürzt IC). Jeder Chip enthält winzige elektronische Schaltungen, in denen die Impulsfolgen verarbeitet werden, die im Computer Anweisungen im Binärcode darstellen. Es gibt Chips für verschiedene Aufgaben. Was die einzelnen Chips für Aufgaben erfüllen, zeigen die Bilder unten.

Der Taktgeber

Mit einem Quarzkristall werden Millionen von Impulsen pro Sekunde erzeugt. Sie regulieren die Impulsströme im Computer .



Der Mikroprozessor

Dieser Chip enthält die Zentraleinheit des Computers (englisch **C**entral **P**rocessing **U**nit oder abgekürzt CPU). Sie erledigt die Hauptarbeit im Computer, rechnet, vergleicht Daten, fällt Entscheidungen und steuert alle anderen Vorgänge. Was sie zu tun hat, erfährt die Zentraleinheit durch Informationen aus dem ROM.

Auf den Mikroprozessor kommt es an!

Mikroprozessoren gibt es in verschiedenen Ausführungen: Verbreitet sind der Z80 (in Computern von Sinclair) und der 6502 (unter anderem im VC-20 und im Oric). Andere Beispiele sind der 6809 im Dragon oder der 9900 im TI-99/4.

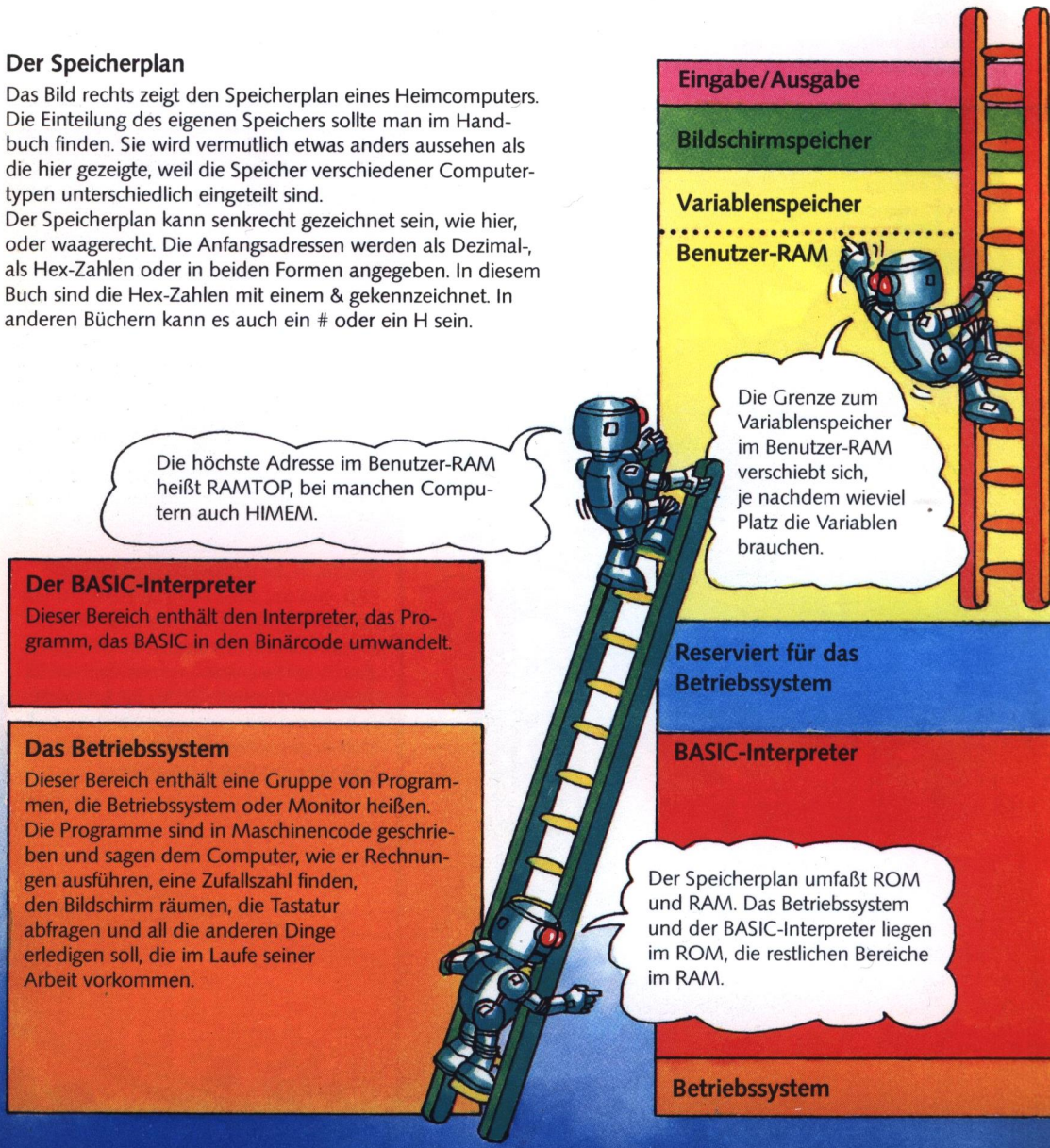
Die verschiedenen Mikroprozessoren arbeiten mit unterschiedlichen Maschinencode-Befehlen, aber alle Computer mit dem gleichen Mikroprozessor verstehen den gleichen Code. Es gibt auch Unterarten der Chips. Beispielsweise ist der Z80A die schnellere Version des Z80, und der 6502A sowie der 6510 (im Commodore 64) stammen vom 6502 ab. Für diese Arten kann man den gleichen Maschinencode verwenden wie für die Originalversion. Dieses Buch bezieht sich speziell auf Computer mit den Mikroprozessoren, die den Code des Z80 oder des 6502 verwenden.

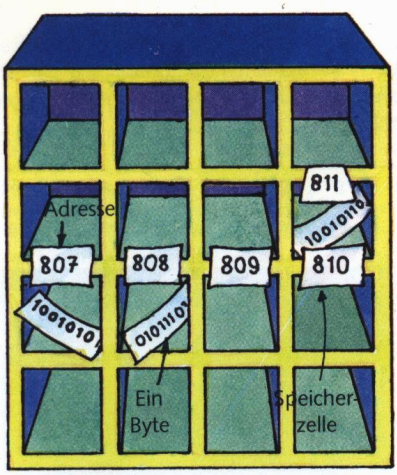
Der Speicher des Computers

Vereinfacht kann man sich den Speicher eines Computers als eine Menge von kleinen Kästchen vorstellen, in die je ein Byte paßt, also eine Anweisung oder Information in Maschinencode. Die Kästchen nennt man *Speicherzellen*, und jede Zelle hat eine Nummer, die *Adresse*, damit der Computer jede einzeln im Speicher finden kann.
Informationen für verschiedene Aufgaben werden in verschiedenen Speicherbereichen untergebracht. Eine Zeichnung, die diese Bereiche samt ihren Anfangsadressen wiedergibt, heißt *Speicherplan* (oder englisch memory map).
In einem Maschinencode-Programm muß man dem Computer sagen, in welchen Zellen er Informationen speichern soll. Das tut man, indem man ihm die Adressen der Zellen gibt. Auch das Programm selbst muß irgendwo gespeichert werden, und dazu muß man die Einteilung des eigenen Speichers kennen.

Der Speicherplan

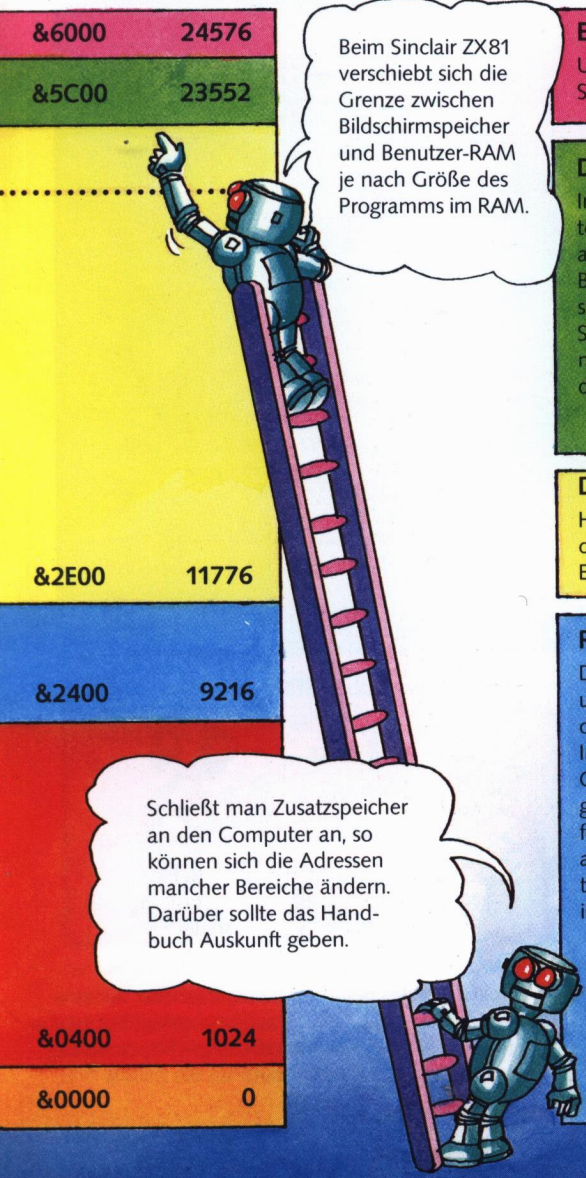
Das Bild rechts zeigt den Speicherplan eines Heimcomputers. Die Einteilung des eigenen Speichers sollte man im Handbuch finden. Sie wird vermutlich etwas anders aussehen als die hier gezeigte, weil die Speicher verschiedener Computertypen unterschiedlich eingeteilt sind.
Der Speicherplan kann senkrecht gezeichnet sein, wie hier, oder waagerecht. Die Anfangsadressen werden als Dezimal-, als Hex-Zahlen oder in beiden Formen angegeben. In diesem Buch sind die Hex-Zahlen mit einem & gekennzeichnet. In anderen Büchern kann es auch ein # oder ein H sein.





Speicheradressen

Viele Computer stellen die Speicheradressen intern mit zwei Bytes dar, also mit 16 Impulsen und Nicht-Impulsen oder Bits. Der größte Speicher, den ein Computer mit einem Z80- oder 6502-Mikroprozessor haben kann, umfaßt 64 KB (ROM und RAM zusammen). Denn die größte Zahl, die man mit 16 binären Ziffern schreiben kann, ist 65 535. Das ist somit die größte Adresse und ergibt 65 536 Speicherzellen, numeriert von 0 bis 65 535. Jede Zelle faßt ein Byte, 1024 Bytes ergeben ein Kilobyte (KB), und 65 536 Bytes ergeben demnach 64 KB (65 536 : 1 024 = 64).



Eingabe/Ausgabe

Unter diesen Adressen erreicht der Computer die Stecker für Eingabe/Ausgabe.

Der Bildschirmspeicher

In diesem Bereich ist gespeichert, was der Computer auf dem Bildschirm anzeigt. Alle Informationen aus diesem Speicher werden automatisch auf den Bildschirm gebracht. Bei vielen Computern entspricht eine Speicherzelle hier einer bestimmten Stelle auf dem Bildschirm. So eine Anzeige nennt man speicherorientiert (englisch memory mapped display).

Das Benutzer-RAM

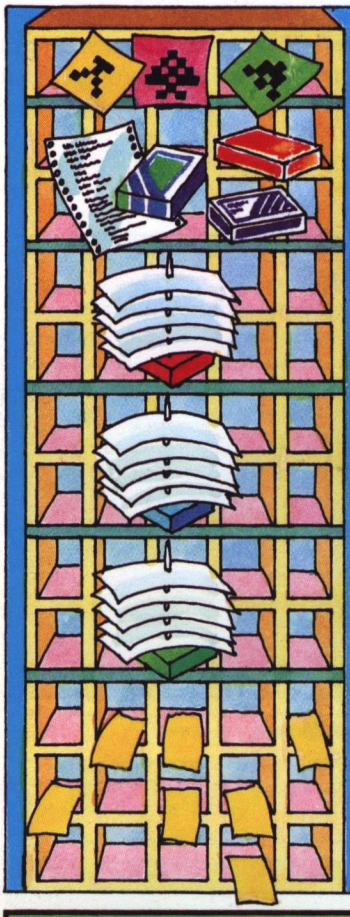
Hier werden die eingegebenen Programme gespeichert. Der Bereich für Variablen liegt am oberen Ende.

Reserviert für das Betriebssystem

Diese Speicherzellen benutzt der Computer, um während der Programmausführung auf dem laufenden zu bleiben. Er speichert hier Informationen etwa über die Position des Cursors, über die Bildschirmfarbe, die zuletzt gedrückte Taste oder die Nummer der laufenden Programmzeile. Mehr darüber steht auf der nächsten Seite. Für manche Computer ist die genaue Einteilung dieses Bereichs im Handbuch angegeben.

Im Arbeitsbereich des Computers

Die Abbildung unten zeigt den Inhalt des Bereichs, der für das Betriebssystem reserviert ist. Wie das bei Ihrem eigenen Computer aussieht, finden Sie möglicherweise im Handbuch. Bei manchen Computern, z. B. denen von Sinclair, liegen die Speicherzellen, die das Betriebssystem beansprucht, nicht in einem zusammenhängenden Bereich, sondern sind über verschiedene Stellen des Speichers verteilt.



Vom Benutzer definierte Grafikzeichen
Wenn man selbst Grafikzeichen definiert, werden sie hier gespeichert.

Puffer
Dies sind Bereiche zur vorübergehenden Speicherung von Daten, die von der Tastatur kommen oder zum Drucker oder zu einer Kassette geschickt werden.

Maschinen-Stapelspeicher (oder Prozessor-Stapelspeicher)
Die Zentraleinheit speichert hier Adressen, die sie sich bei der Bearbeitung von Maschinencode-Programmen „merken" muß.

BASIC-Stapelspeicher (oder GOSUB-Stapelspeicher)
Hier werden Zeilennummern für die BASIC-Anweisungen GOSUB und RETURN gespeichert.

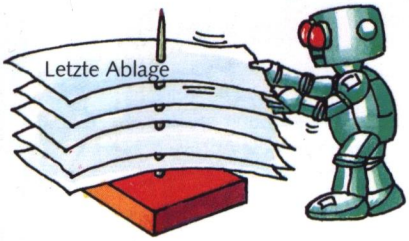
Rechner-Stapelspeicher
Zahlen, die die Zentraleinheit für ihre Berechnungen braucht, werden hier vorübergehend gespeichert.

Systemvariablen
Dies ist eine Reihe von Speicherzellen, in denen die Zentraleinheit Informationen über die Vorgänge im Computer ablegt. Das kann z. B. die augenblickliche Position des Cursors auf dem Bildschirm sein, der Code der zuletzt gedrückten Taste oder die Anfangsadresse des Variablenspeichers.

Speicherseiten

Man kann sich den Speicher wie ein Buch in „Seiten" aufgeteilt vorstellen. Bei Heimcomputern umfaßt eine Seite gewöhnlich 256 Speicherplätze. Vier Seiten ergeben dann ein Kilobyte ($4 \cdot 256 = 1024$).
Die Zellen 0 bis 255 werden manchmal Null-Seite (zero page) genannt. Oft beginnen die Speicherbereiche auf einer neuen Seite. Das Benutzer-RAM etwa (siehe vorige Seite) beginnt auf Seite 45, wobei die erste Seite als Null-Seite gezählt wird.

Noch etwas über Stapelspeicher



Der Computer benutzt die Stapelspeicher auf eine bestimmte Weise: Was er zuletzt abgelegt hat, holt er zuerst wieder ab. Das nennt sich LIFO-Speicherung (von englisch **l**ast **i**n, **f**irst **o**ut).

Hexadezimalzahlen

In Maschinencode-Programmen schreibt man Zahlen und Adressen immer in Hex. Unten wird erklärt, wie man Dezimalzahlen in Hexadezimalzahlen umwandelt und umgekehrt.

Dezimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadezimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Die Tabelle zeigt die Hex-Ziffern (0 – 9 und A – F) und ihre Dezimalwerte. Für Zahlen, die größer sind als 15 (F), benutzt man mehrere Ziffern, genauso wie bei Dezimalzahlen, die größer sind als 9. Die Stellung in der Zahl bestimmt den Wert jeder Ziffer.

Dezimal			
1 000er	100er	10er	1er
1	2	2	6

Hexadezimal		
256er	16er	1er
4	C	A

Bei Dezimalzahlen gibt die erste Ziffer von rechts die 1er an, die nächste Ziffer die 10er, die dritte Ziffer die 100er (10^2) usw.

Bei Hex-Zahlen gibt die erste Ziffer von rechts auch die 1er an, die nächste Ziffer aber die 16er, die dritte Ziffer die 256er (16^2) usw.

16^2 16^1 16^0

Umwandlung Hexadezimal/Dezimal

Für die Umwandlung, z. B. von 4CA, sucht man den Dezimalwert für jede Ziffer, multipliziert ihn mit dem jeweiligen Stellenwert und addiert die Ergebnisse.

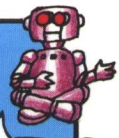
256er	16er	1er
4	C	A

4 12 10 Dezimalwert

· 256 · 16 · 1

1 024 + 192 + 10 = 1 226

&4CA ist dezimal 1226.



Versuchen Sie, &A7 in Dezimal und 513 in Hex umzuwandeln. (Lösung auf Seite 46.)

Umwandlung Dezimal/Hexadezimal

Für die Umwandlung, z. B. von 1 226, teilt man die Zahl zuerst durch 256. Dann weiß man, wie viele 256er enthalten sind. Den Rest teilt man durch 16, um die Anzahl der 16er zu ermitteln; was jetzt bleibt, sind die 1er. Zuletzt wandelt man die Ergebnisse in Hex-Ziffern um.*

1226 : 256 = 4 . . . . . 4 ist 4 in Hex.
Rest 202

202 : 16 = 12 . . . . . 12 ist C in Hex.
Rest 10. . . . . 10 ist A in Hex.

1 226 ist 4CA in Hex.

Umwandlung von Hex-Adressen

In einer Hex-Adresse, z. B. 5C64, geben die ersten beiden Ziffern von links die Speicherseite (siehe vorige Seite) an, die beiden Ziffern rechts die Stelle auf der Seite.

Hexadezimal in Dezimal

Adresse in Hexadezimal: &5C64

Seitenzahl = &5C = 92 in Dezimal
Stelle auf Seite = &64 = 100 in Dezimal

92 · 256 = 23 552 + 100
= 23 652

Die Adresse 5C64 in Hex ist dezimal 23 652.

Zur Umwandlung einer Hex-Adresse in eine Dezimaladresse wandelt man zuerst beide Ziffernpaare in Dezimalzahlen um. Dann multipliziert man die Seitenzahl mit 256 – jede Seite hat 256 Speicherzellen – und addiert die Zahl für die Stelle auf der Seite.

Dezimal in Hexadezimal

Dezimaladresse: 23 652

23 652 : 256 = 92 (Speicherseite)
Rest 100 (Stelle auf der Seite)

92 : 16 = 5 Rest 12 = &5C
100 : 16 = 6 Rest 4 = &64

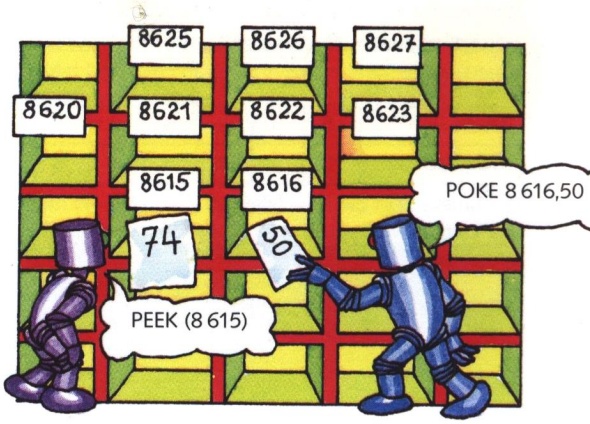
Die Dezimaladresse 23 652 ist 5C64 in Hex.

Zur Umwandlung einer Dezimaladresse in eine Hexadezimaladresse teilt man sie durch 256, um die Seitenzahl zu finden. Der Rest ergibt die Stelle auf der Seite. Dann wandelt man die Zahlen in Hex-Ziffern um, wie es oben gezeigt wird.

* Wie man das mit einem Taschenrechner macht, steht auf Seite 41.

PEEK und POKE

In BASIC ermöglichen es die beiden Anweisungen PEEK und POKE*, sich die Bytes anzusehen, die an den entsprechenden Speicherplätzen abgelegt sind, und sie zu ändern. Oft gibt man die Speicheradresse, die man erreichen will, als Dezimalzahl an, bei manchen Computern auch als Hex-Zahl. Bei Hex-Zahlen muß man bei der Eingabe vor die Zahl ein Zeichen wie & oder # setzen. Da das von Computer zu Computer verschieden ist, sollte man im Handbuch nachschlagen, wie man vorgehen muß.



Mit PEEK kann man sich alle Bytes im Speicher des Computers anschauen. Mit POKE lassen sich nur die Bytes im RAM ändern. Der Inhalt des ROM ist nicht veränderbar.

So benutzt man PEEK

```
PRINT PEEK(12345)
46
PRINT PEEK(720)
240
PRINT PEEK(8643)
0
LET A=PEEK(1024)
PRINT A
176
```

```
10 FOR J=700 TO 725
20 PRINT PEEK(J);",";
30 NEXT J
```

```
RUN
36,27,234,56,21,0,0,
0,0,45,32,67,121,45,
47,89,63,21,0,87,241,
202,225,63,87,16,
```

Dies sind die entsprechenden Dezimalwerte der Bytes des Maschinencodes.

Wenn Sie sich eine Speicherzelle ansehen wollen, benutzen Sie die Anweisung PEEK (oder die entsprechende für Ihren Computer) mit der zugehörigen Adresse. PRINT PEEK zeigt das Ergebnis auf dem Bildschirm, mit LET speichert man es in einer Variablen und holt diese auf den Bildschirm (siehe oben links).

Ein kleines Programm mit einer FOR/NEXT-Schleife, wie das Programm oben in der Mitte, zeigt die Bytes in einer Reihe von Speicherzellen. So können Sie sich ansehen, was in den Bereichen gespeichert ist, die in Ihrem Speicherplan angegeben sind.

So benutzt man POKE

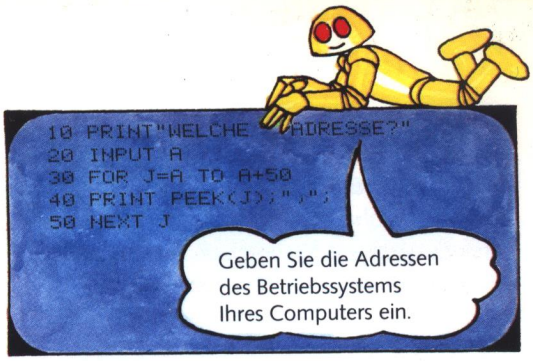
Die Bilder oben zeigen, wie man POKE verwendet. Die Adresse muß im RAM liegen. Schreibt man allerdings einen neuen Wert in eine Zelle, die vom Betriebssystem benutzt wird, so kann der Computer „aussteigen": Nachdem man ihn einmal aus- und wieder eingeschaltet hat, arbeitet er wieder normal.

Probieren Sie einmal das Programm oben aus: Damit können Sie mehrere Zahlen in eine Reihe von Speicherzellen einschreiben.
Die Zahlen müssen zwischen 0 und 255 liegen. Das sind die Werte, die sich mit acht Binärziffern (ein Byte im Computercode) darstellen lassen.

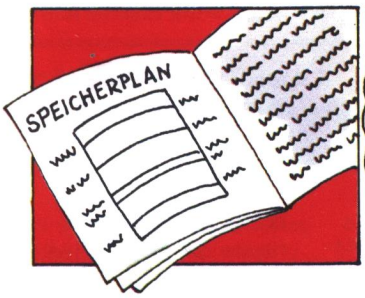
* Einige Computer verwenden andere Begriffe. Welche, steht in Ihrem Handbuch.

Die „Hausnummern" der Speicherzellen

Weist man den Computer an, den Inhalt einer Speicherzelle zu zeigen, so erhält man immer eine Dezimalzahl zwischen 0 und 255. Denn jede Speicherzelle enthält ein Byte, und der größte Wert, der sich mit acht binären Ziffern darstellen läßt, ist 255. Es gibt also nur 256 verschiedene Bytes (0 bis 255) im Computercode, und jedes Byte kann mehrere Bedeutungen für den Computer haben.
Beispielsweise kann die Binärzahl 00110000 (dezimal 48) der Code für eine Anweisung, für ein Zeichen auf der Tastatur oder Teil der Adresse einer Speicherzelle sein. (Jede Adresse besteht aus zwei Bytes.)

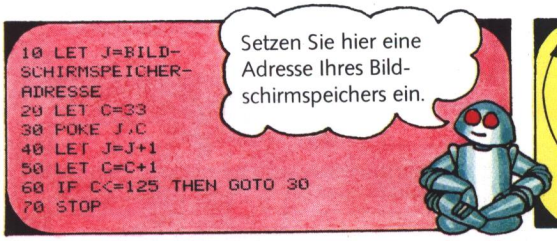


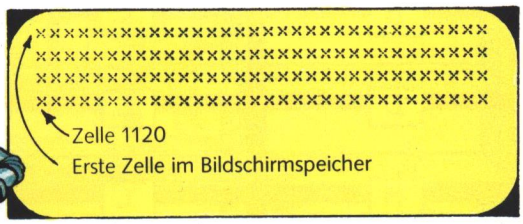
Im Handbuch sind die Adressen verzeichnet, unter denen das Betriebssystem im ROM gespeichert ist. Sieht man sich die Bytes an, die man unter diesen Adressen findet, so erhält man Dezimalzahlen auf dem Bildschirm. Es sind die Dezimalwerte der Bytes eines Maschinencode-Programms des Betriebssystems.



Schlagen Sie im Handbuch die Adressen Ihres Bildschirmspeichers nach und schreiben Sie dort mit POKE Zahlen hinein. Das Ergebnis erscheint ohne PRINT-Anweisung, weil die Bytes in diesem Bereich vom Computer als Codes für Zeichen interpretiert und automatisch angezeigt werden.*

Viele Computer benutzen den ASCII-Code, um Zeichen zu speichern. Jedem Zeichen (Buchstabe, Ziffer, Grafikzeichen usw.) entspricht ein Byte mit einem bestimmten Wert. Einige Computer, wie der Sinclair ZX81, ordnen den Werten allerdings andere Zeichen zu. Im Handbuch steht, welchen Code Ihr Computer benutzt.





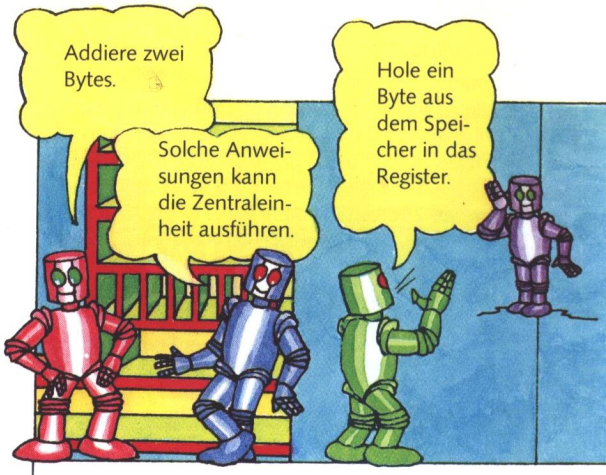
Mit einem solchen Programm kann man sich den Zeichenvorrat des Computers zeigen lassen, wenn er den ASCII-Code benutzt. Sonst muß man in Zeile 20 einen anderen Anfangswert für C und in Zeile 60 einen anderen Endwert nehmen. Bei manchen Computern ist der Bildschirmspeicher auch so aufgebaut, daß das Programm nicht läuft. Hier kann man experimentieren.

Bei den meisten Computern kann man sich ein Zeichen an einer bestimmten Stelle des Bildschirms zeigen lassen, wenn man die Adresse für diese Stelle ausrechnet. Beginnt etwa der Bildschirmspeicher bei 1 024 und der Computer schreibt 32 Zeichen pro Zeile, dann ist die Adresse des ersten Zeichens der vierten Zeile $1024 + (32 \cdot 3) = 1120$.

* Beim Sinclair ZX81 geht das nur mit Zusatzspeicher, und der Spectrum speichert Informationen für eine Stelle des Bildschirms in verschiedenen Zellen, so daß POKE hier nicht in gewohnter Weise funktioniert.

Blick in die Zentraleinheit

Wenn der Computer arbeitet, holt er sich Bytes für Anweisungen und Daten aus dem Speicher und führt die Anweisungen dann in der Zentraleinheit aus.
Die Zentraleinheit besteht aus drei Hauptteilen: den *Registern* zum Speichern von Daten, die gerade bearbeitet werden, der *Arithmetisch-logischen Einheit* (ALU) zum Addieren, Subtrahieren und Vergleichen von Bytes und aus der *Steuereinheit*, die diese Tätigkeiten koordiniert.
Die Anordnung der Register ist bei den Mikroprozessoren Z80 und 6502 unterschiedlich, wie die Abbildungen unten zeigen.



Diese Bilder zeigen Anweisungen, die die Zentraleinheit ausführen kann. Sie sind alle sehr einfach. Die Zentraleinheit kann Bytes aus dem Speicher in ein Register holen, Bytes von einem Register in ein anderes übertragen, in der ALU bearbeiten und das Ergebnis im Speicher ablegen. Selbst einfache Auf-

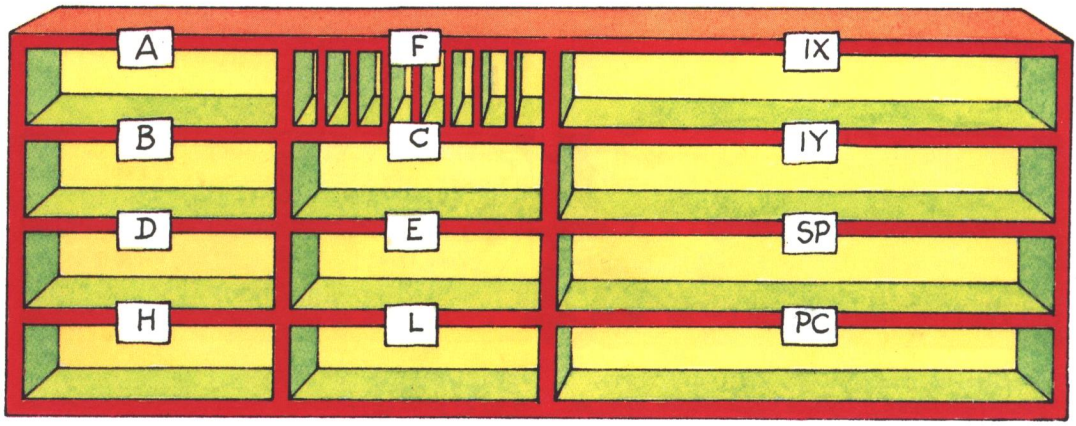
Die Register des Z80

Der Hauptunterschied zwischen Z80 und 6502 ist, daß der Z80 mehr Register hat. Das bedeutet, daß in der Zentraleinheit mehr Bytes zeitweilig gespeichert werden können, während der 6502 sie zum Speicher zurückschicken muß.

A bedeutet *Akkumulator* (abgekürzt Akku). Er ist das wichtigste Register und speichert Bytes auf dem Weg von oder zur Arithmetisch-logischen Einheit. Er kann immer nur ein Byte speichern.

F ist das *Kennzeichen-* oder *Flagregister*. Es enthält acht Bits. (Zwei davon werden nicht benutzt.) Jedes Bit stellt ein Signal dar. Das Übertragskennzeichen (oder Carry Flag) z. B. wird auf 1 gesetzt, wenn ein Ergebnis größer ist als 255 und daher nicht in einem Byte Platz hat.

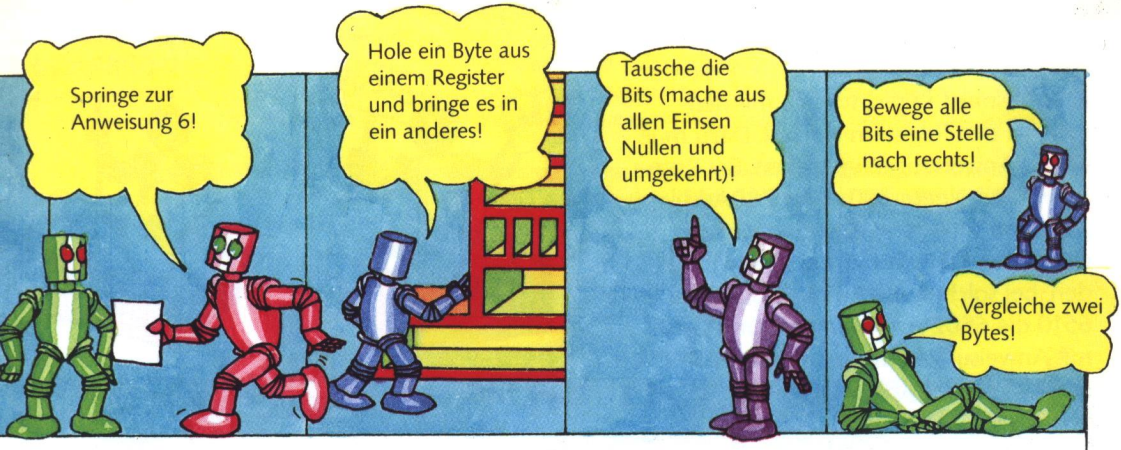
IX und **IY** sind die *Indexregister*. Sie umfassen 16 Bits und werden bei bestimmten Anweisungen benutzt, mit denen man Adressen berechnet.



B, **C**, **D**, **E**, **H** und **L** sind *Universalregister* zum Speichern von je einem Byte. Sie können auch zu Paaren zusammengefaßt werden (BC, DE und HL), um zwei Bytes zu speichern.

SP ist der *Stapelzeiger* (englisch **S**tack **P**ointer). Dieses 16-Bit-Register enthält die Adresse der letzten Ablage auf dem Maschinen-Stapelspeicher (siehe Seite 10).

PC ist der *Programmzähler* (englisch **P**rogram **C**ounter). Hier ist die 16-Bit-Adresse des Bytes gespeichert, das als nächstes aus dem Speicher geholt wird. Danach wird die Adresse jedesmal um 1 erhöht.



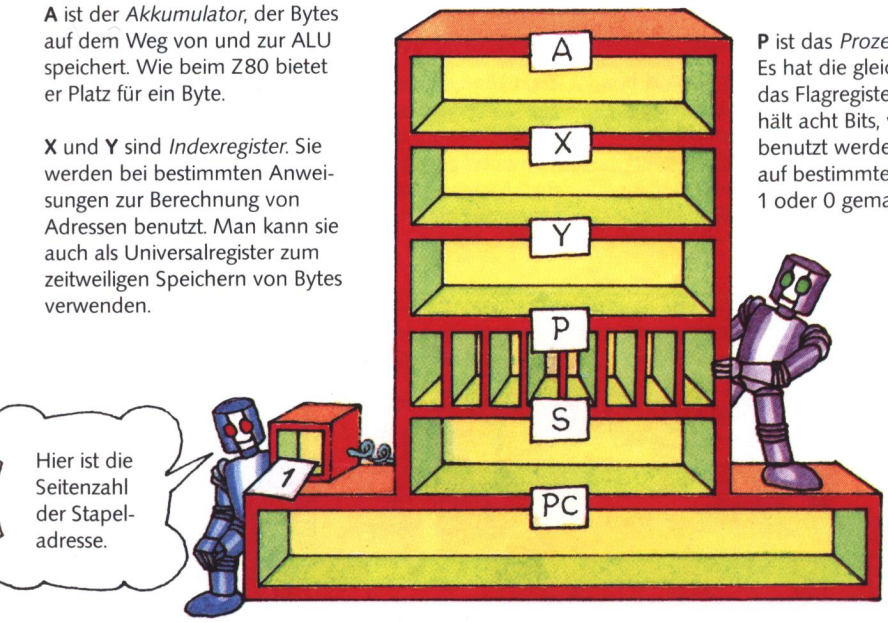
gaben, wie zwei Zahlen zu addieren und auf dem Bildschirm zu zeigen, erfordert über hundert Anweisungen. Die Zentraleinheit kann mehr als eine halbe Million pro Sekunde ausführen.
Für jede Operation holt die Zentraleinheit ein Anweisungsbyte aus dem ROM oder RAM, speichert ein oder mehrere Datenbytes in Registern und führt dann die geforderte Operation aus. Mit dem Maschinencode weist man die Zentraleinheit zu bestimmten Operationen an, die sie mit Hilfe der ALU und der Steuereinheit automatisch ausführt. Wie sie das macht, läßt sich von außen nicht beeinflussen.

Die Register des 6502

Die Hauptregister des 6502 sind die gleichen wie die des Z80. Einige von ihnen haben allerdings andere Namen.

A ist der *Akkumulator*, der Bytes auf dem Weg von und zur ALU speichert. Wie beim Z80 bietet er Platz für ein Byte.

X und **Y** sind *Indexregister*. Sie werden bei bestimmten Anweisungen zur Berechnung von Adressen benutzt. Man kann sie auch als Universalregister zum zeitweiligen Speichern von Bytes verwenden.



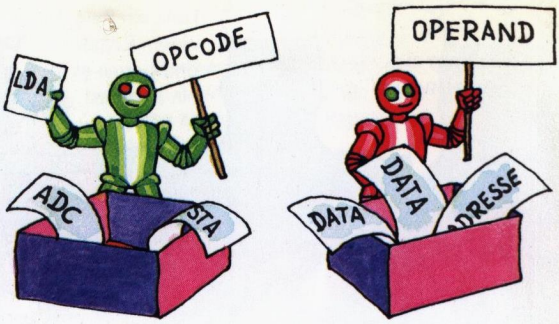
P ist das *Prozessor-Statusregister*. Es hat die gleiche Aufgabe wie das Flagregister des Z80. Es enthält acht Bits, von denen sieben benutzt werden. Die Bits werden auf bestimmte Ereignisse hin zu 1 oder 0 gemacht.

PC ist der *Programmzähler* wie beim Z80.

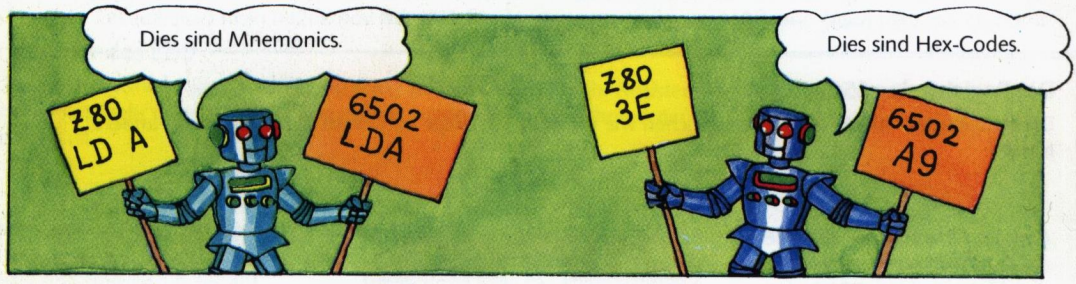
S ist der *Stapelzeiger*. Er enthält die Adresse der letzten Ablage auf dem Stapel; das ist der RAM-Bereich, in dem die Zentraleinheit sich Daten „merkt". Da der Stapelzeiger des 6502 nur ein Byte enthält, findet man dort nur den Teil der 16-Bit-Adresse, der auf die Stelle auf der einen Speicherseite zeigt. Die Seitenzahl steht im Stapelzeiger-Register und ist immer 1. Der Stapel liegt also immer auf Seite 1.

Anweisungen für die Zentraleinheit

Ein Maschinencode-Programm besteht aus einer Liste von Anweisungen, die der Zentraleinheit sagen, was sie mit den Bytes in den Registern tun soll. Dazu muß sie die Anweisungen verstehen. Bei Computern mit einem Z80- oder Z80A-Mikroprozessor muß man Anweisungen aus dem Anweisungsvorrat des Z80 verwenden, bei solchen mit 6502, 6502A und 6510 die 6502-Anweisungen. Eine Liste mit Anweisungen für den Z80 und den 6502 finden Sie am Ende des Buches.

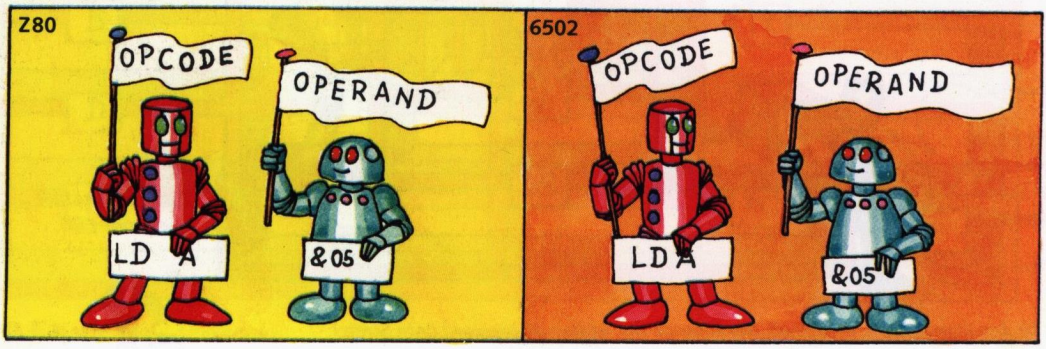


Die meisten Maschinencode-Anweisungen bestehen aus zwei Teilen: aus dem *Opcode* (Operationscode) und dem *Operand*. Der Opcode sagt der Zentraleinheit, was sie zu tun hat, und der Operand sagt, wo die Daten zu finden sind, die bearbeitet werden sollen. Jeder Opcode ist eine Anweisung aus dem Anweisungsvorrat.



Opcodes kann man als Mnemonics schreiben, d. h. als Abkürzung für die Operation, oder als Hex-Zahl, die dem Binärcode des Computers entspricht. Zum Beispiel sind LD A (für den Z80) und LDA (für den 6502) Mnemonics und bedeuten „hole ein Byte in den Akkumulator". Das gleiche in Hex ist 3E für den Z80 und A9 für den 6502.

Zwar kann man sich Mnemonics leichter merken als Hex-Zahlen, aber man braucht einen Assembler, also ein Übersetzungsprogramm, damit man sie in den Computer eingeben kann*. Ohne Assembler schreibt man Maschinencode-Programme am einfachsten in Mnemonics und übersetzt diese vor dem Eintippen selbst in Hex.



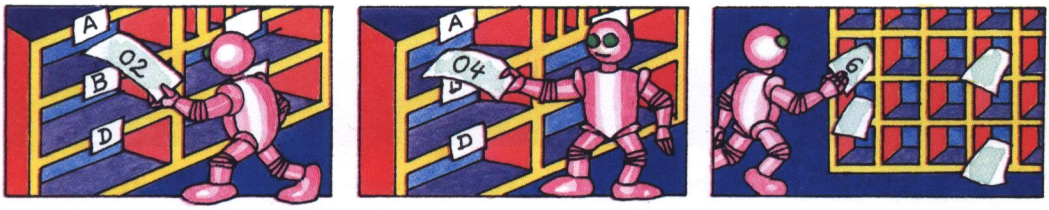
Hier sind zwei Maschinencode-Anweisungen als Mnemonics dargestellt, eine für den Z80 und eine für den 6502. Beide sagen der Zentraleinheit, daß sie &05 (& bedeutet hier Hex) im Akkumulator speichern soll. Bei den Mnemonics des 6502 zeigt ein # vor den Hex-Zahlen, daß sie Bestandteile von Daten sind.

*Über Assembler steht mehr auf Seite 40.

Ein einfaches Programm

Hier sieht man zwei Programme, mit denen zwei Zahlen addiert werden sollen, ein Programm für den Z80 und eines für den 6502. Beide Programme sind mit Mnemonics geschrieben. Streng genommen nennt man Programme in Mnemonics Assemblersprachen-Programme und solche in Hex-Codes Maschinencode-Programme. Auf der nächsten Seite findet man Hinweise, wie man die Programme in Maschinencode übersetzt, und auf den darauffolgenden Seiten, wie man ein Programm in den eigenen Computer laden und ablaufen lassen kann.

Die Programme für den Z80 und 6502 enthalten die gleichen Schritte, obwohl die eigentlichen Anweisungen unterschiedlich sind*. Der 6502 kann nur mit Daten arbeiten, die sich im Akkumulator befinden, der Z80 kann für große Zahlen auch das Registerpaar HL verwenden.



Zum Addieren holt man zunächst die erste Zahl in den Akkumulator. Dann addiert man die zweite Zahl. Das Ergebnis steht im Akkumulator und wird in den Speicher gebracht. Die Mnemonics für die Opcodes dieser Anweisungen sind unten aufgeführt.

Z80-Mnemonics	**Bedeutung**
LD A, Zahl	Lade A mit einer Zahl. A heißt Akkumulator und LD ist kurz für englisch **load** (= lade).
ADD A, Zahl	Addiere eine Zahl zu A (Akkumulator).
LD (Adresse), A	Speichere den Inhalt von A unter einer bestimmten Adresse. (Adressen werden in Klammern geschrieben.)

Opcode und Operand werden beim Z80 durch Kommas getrennt.

6502-Mnemonics	**Bedeutung**
LDA Zahl	Lade A mit einer Zahl. A heißt Akkumulator, und LD ist kurz für englisch **load** (= lade).
ADC Zahl	ADC ist das Mnemonic für englisch **add** with **c**arry (= addiere mit Übertrag). Dabei wird nicht nur die Zahl zum Inhalt des Akkumulators addiert, sondern auch der Inhalt des Übertragskennzeichens. (Mehr darüber auf Seite 29.)
STA Adresse	Speichere A (Inhalt des Akkumulators) unter einer bestimmten Adresse. ST ist kurz für englisch **st**ore (= speichere).

Z80-Addierprogramm

```
LD A, &02
ADD A, &04
LD (&7F57), A
```

Daten

Adresse

Dieses Programm braucht drei Opcodes: LD A, ADD A und LD (Adresse), A.

6502-Addierprogramm

```
LDA #&02
ADC #&04
STA &7F57
```

Daten

Adresse

Das # bedeutet, daß der Operand ein Datum und keine Adresse ist.

Nun können Sie Daten und Adressen einspeichern. In diesen Beispielen addieren die Programme 2 Hex und 4 Hex (das gleiche wie 2 und 4 Dezimal) und speichern das Ergebnis unter der Hex-Adresse 7F57.

* Von hier an können Sie die Programme, die nicht für die Zentraleinheit *Ihres* Computers (Z80 oder 6502) geeignet sind, überspringen.

Wie man ein Programm in Hex übersetzt

Zur Übersetzung der Mnemonics in Hex-Codes muß man in einer Liste nachschlagen. Eine solche Liste mit Mnemonics und Hex-Codes befindet sich am Ende des Buches. Man muß dabei allerdings gut aufpassen, da es für jede Anweisung verschiedene Hex-Codes gibt, je nachdem ob es sich bei dem Operanden um Daten, Adressen oder um ein Register handelt. Als Beispiele folgen hier verschiedene Opcodes mit ihren Hex-Codes zum Laden des Akkumulators.

Z80		6502	
Mnemonics	**Hex-Codes**	**Mnemonics**	**Hex-Codes**
LD A, Datum	3E, Datum	LDA Datum	A9 Datum
LD A, (Adresse)	3A, Adresse	LDA Adresse	AD Adresse

Ist der Operand ein Datum, dann nennt man das *unmittelbare Adressierung*; ist er eine Adresse, dann handelt es sich um *direkte Adressierung*. Alle Anweisungen, die im Buch behandelt werden, stehen in der Liste mit Mnemonics und Hex-Codes am Ende des Buches. Hinweise auf Bücher mit vollständigen Listen finden Sie auf Seite 40.

Z80-Addierprogramm		6502-Addierprogramm	
Mnemonics	**Hex-Codes**	**Mnemonics**	**Hex-Codes**
LD A, Datum	3E, Datum	LDA Datum	A9 Datum
ADD A, Datum	C6, Datum	ADC Datum	69 Datum
LD (Adresse), A	32, Adresse	STA Adresse	8D Adresse

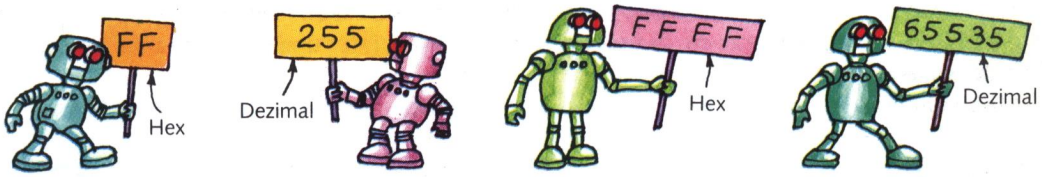
Dies sind die Hex-Codes der Addierprogramme für den Z80 und den 6502. Anweisungen in Mnemonics (Assemblersprache) nennt man auch *Quellcode*, Anweisungen in Hex-Code (Maschinensprache) heißen *Objektcode*.

Z80-Addierprogramm mit Daten		6502-Addierprogramm mit Daten	
Mnemonics	**Hex-Codes**	**Mnemonics**	**Hex-Codes**
LD A, &02	3E, 02	LDA #&02	A9 02
ADD A, &04	C6, 04	ADC #&04	69 04
LD (&7F57), A	32, 577F	STA &7F57	8D 577F

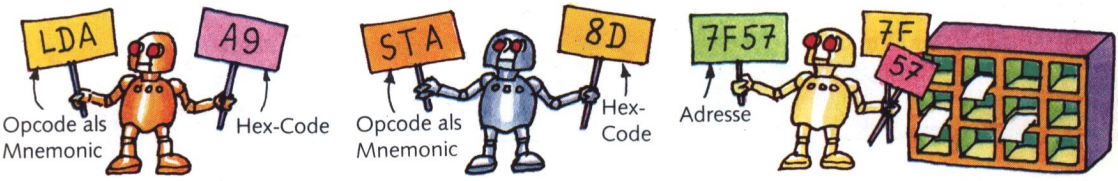
Das Eintragen der Daten ist recht einfach. Allerdings muß man bei Adressen im Maschinencode die beiden Ziffernpaare vertauschen, die die Adresse bilden. Mehr darüber steht auf der nächsten Seite.

Noch etwas über Hex-Codes

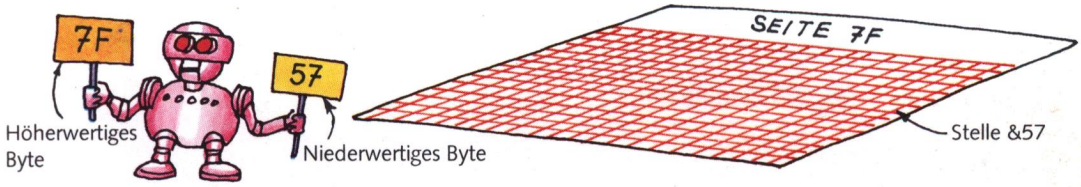
Maschinencode-Programme schreibt man besser in Hex als in Dezimalzahlen, weil sich der binäre Computercode leichter in Hex-Zahlen als in Dezimalzahlen umrechnen läßt.



Die größte 16-Bit-Adresse ist z. B. dezimal 65 535, in Hex dagegen FFFF, und die größte Zahl mit einem Byte (acht Binärziffern) ist dezimal 255, in Hex aber FF*.



Die meisten Opcodes im Anweisungsvorrat eines Computers sind ein Byte lang, in Hex also zwei Ziffern. Adressen brauchen jedoch zwei Bytes, in Hex also zwei Ziffernpaare.



Das linke Hex-Ziffernpaar nennt man *höherwertiges Byte*; es gibt die Speicherseite (siehe Seite 10) an, auf der die Adresse zu finden ist. Das rechte Ziffernpaar heißt *niederwertiges Byte* und bezeichnet die Stelle der Speicherzelle auf der Seite (eine Seite = 256 Speicherzellen). Die Zentraleinheit braucht die Angabe der Stelle auf der Seite immer *vor* der Seitenzahl. Darum muß man im Maschinencode die Ziffernpaare vertauschen.

So liest man Maschinencode-Programme

Maschinencode-Programme in Zeitschriften sehen sehr verwirrend aus, wenn man sich nicht klar ist, wie sie dargestellt werden. Unten finden Sie zwei Beispiele, wie Maschinenprogramme gedruckt sein können. (Diese Programme sind unvollständig und würden auf dem Computer nicht laufen.)

Hex-Speicherauszug

Hex-Adresse

Jedes Ziffernpaar ist eine Anweisung, ein Datum oder eine Adressenhälfte.

```
3A30  00 00 00 00 DB 01 CB 1F
3A38  CB 1F 30 07 3E 10 D3 00
3A40  CF 37 3F 21 2F 39 11 00
3A48  D0 01 C0 03 C5 0E 27 71
3A50  1A FE 58 28 28 CD C5 3A
3A58  CD A0 3A D3 01 4E 0D 79
3A60  FE 00 FA 88 3A F5 CD BD
3A68  0B FE 64 CA 06 3B F1 71
```

Diese Darstellung nennt man einen *Speicherauszug*. Die ersten vier Ziffern einer Zeile bilden eine Adresse, die restlichen Paare die Hex-Codes für Anweisungen, Daten oder Adressen. Der erste Code in einer Zeile ist in der Adresse am Anfang der Zeile gespeichert, die restlichen Codes jeweils eine Adresse weiter.

Assemblerlisting

Adresse — Hex-Codes — Mnemonics

```
0340   A2 00      LDX  #&00
0342   BD 4E 03   LDA  &034E,X
0345   9D C0 83   STA  &83C0,X
0348   E8         INX
0349   E0 0B      CPX  #&0B
034B   D0 F5      BNE  &F5
034D   00         BRK
```

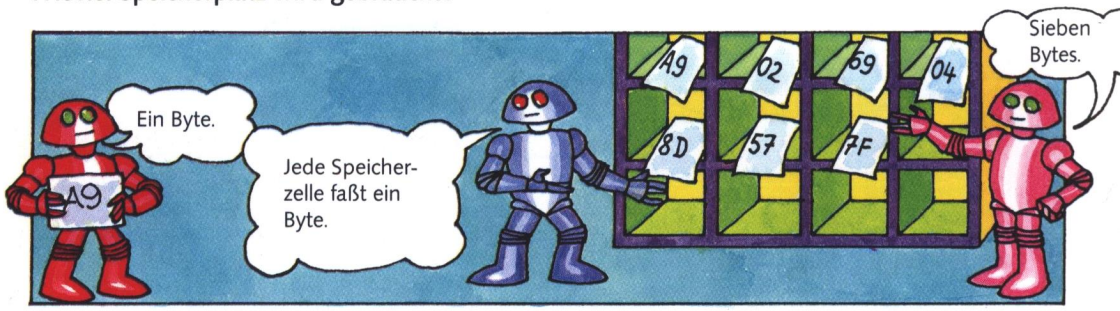
Diese Darstellung ist ein *Assemblerlisting*. Es enthält außer den Hex-Codes auch die Mnemonics. Die erste Zahl in jeder Zeile gibt an, unter welcher Adresse der folgende Opcode gespeichert ist. Dann folgen die Hex-Codes und danach die Mnemonics.

*Wie man Binärzahlen in Dezimalzahlen umrechnet, steht auf Seite 28.

Gesucht: freier Speicherplatz

Bevor man das Addierprogramm von Seite 18 ablaufen lassen kann, gibt es noch einiges zu tun. Zuerst muß man im RAM Platz finden, um das Programm zu speichern. Bei BASIC-Programmen erledigt das der BASIC-Interpreter automatisch. Gibt man dem Computer aber ein Maschinencode-Programm, so umgeht man den Interpreter und muß selbst für Speicherplatz sorgen.
Man muß einen Bereich im RAM suchen, in dem der Maschinencode nicht andere, bereits gespeicherte Informationen stört. Z. B. darf man den Maschinencode nicht in Zellen speichern, die für das Betriebssystem reserviert sind. Tut man das doch, so findet der Computer plötzlich Ihren Maschinencode an Stellen, an denen er für seine Funktion wichtige Informationen gespeichert hatte. Auch in gleichzeitig gespeicherte BASIC-Programme darf man nicht hineingeraten. Die Folge davon wäre, daß das Programm „abstürzt". Der Computer muß dann einmal ausgeschaltet und erneut eingeschaltet werden, damit er wieder arbeitet. Ihr Programm geht dabei allerdings verloren.

Wieviel Speicherplatz wird gebraucht?



Es ist leicht, die Länge eines Maschinencode-Programms zu bestimmen: Man zählt einfach die Paare von Hex-Ziffern: Jedes Paar ist ein Byte. Das Addierprogramm hat z. B. sieben Bytes.

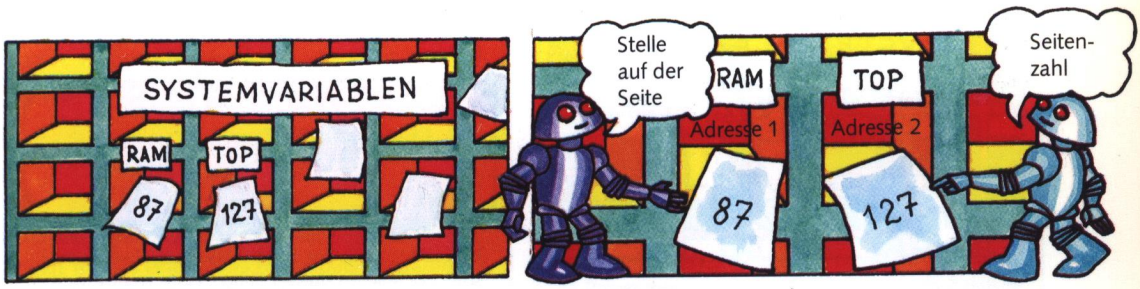
Die meisten Maschinencode-Programme sind recht kurz. Hundert Bytes Platz für den Maschinencode sollte für den Anfang völlig ausreichen.

Speicherplatz für das Maschinencode-Programm

Ein guter Platz für Maschinencode-Programme ist das obere Ende des Benutzer-RAM. Man muß aber dafür sorgen, daß der Computer dort nichts anderes ablegt. Dazu senkt man die Obergrenze des Benutzer-RAM ab (siehe nächste Seite). Dadurch „vergißt" der Computer den Bereich über der Obergrenze und benutzt ihn erst wieder, wenn man ihm sagt, daß er dort Maschinencode speichern soll.
Die Zahl, die der Computer als Obergrenze gespeichert hat, heißt RAMTOP, HIMEM oder einfach RAM-Obergrenze.

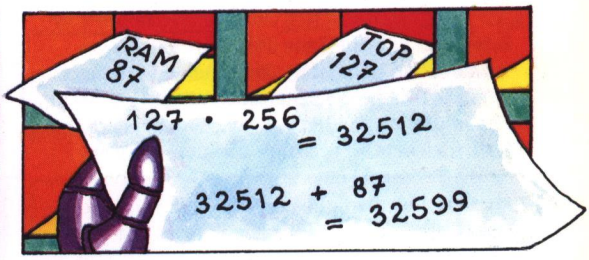
So senkt man die Obergrenze des Benutzer-RAM ab

Die Obergrenze hat der Computer in einer Systemvariablen gespeichert. Den Wert dieser Variablen kann man nachträglich ändern. Die Anweisungen dafür sind von Computer zu Computer verschieden, entsprechen jedoch meist dem unten gezeigten Verfahren. Schauen Sie aber zur Sicherheit im Handbuch Ihres Computers nach, weil es Abweichungen geben kann oder es bei Ihrem Computer sogar einfacher ist, Platz für Maschinencode zu schaffen.



Das RAMTOP ist eine 16-Byte-Adresse, es belegt also zwei Speicherzellen im Bereich für Systemvariablen: eine für die Seitenzahl und eine für die Stelle auf der Seite. Die Adressen dieser beiden Bytes findet man im Handbuch, wenn man unter RAMTOP, HIMEM oder RAM-Obergrenze nachschaut. Dort ist also die höchste RAM-Adresse gespeichert, und zwar zuerst das niederwertige Byte (Stelle auf der Seite) und dann das höherwertige (Seitenzahl).

```
PRINT PEEK (ADRESSE 1) +
256*PEEK (ADRESSE 2)
```

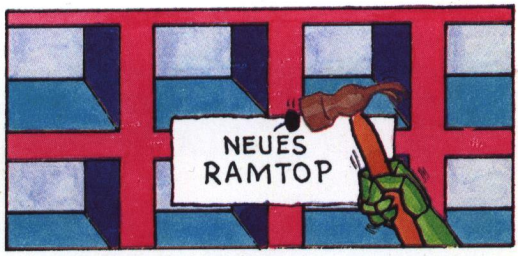


Man kann sich mit PRINT PEEK die Adresse RAMTOP zeigen lassen, wie oben dargestellt. Dazu muß man noch die Adressen der entsprechenden Systemvariablen einfügen.

Dieses Kurzprogramm rechnet die zwei Bytes von RAMTOP in eine Dezimalzahl um. Die Seitenzahl wird mit 256 multipliziert und die Stelle auf der Seite zum Ergebnis addiert.

```
CLEAR ADRESSE RAMTOP - 100
                       Sinclair
                       Spectrum

HIMEM ADRESSE RAMTOP - 100
                       Oric
```

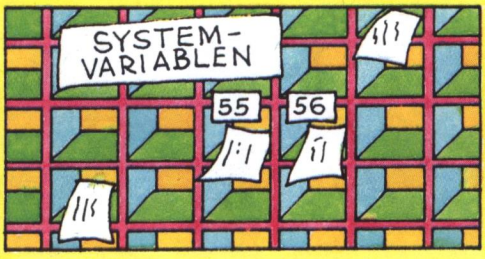


Viele Computer haben spezielle Anweisungen, mit denen man die RAM-Obergrenze ändert. Der Sinclair Spectrum benutzt dazu die Anweisung CLEAR, der Oric HIMEM. Der Anweisung folgt jeweils die höchste RAM-Adresse minus der Anzahl der Bytes, die man für Maschinencode frei haben will*. In diesem Beispiel wird die Obergrenze um 100 Zellen abgesenkt. Damit sind 99 Bytes für Maschinencode frei, ausgehend von der Adresse hinter RAMTOP.

* Auf der nächsten Seite steht, wie man beim VC-20 die Obergrenze absenkt und wo man beim Sinclair ZX81 Maschinencode speichert.

Tips für den VC-20

Der VC-20 hat keine spezielle Anweisung zum Ändern der Adressen in den Systemvariablen. Bei ihm senkt man die Obergrenze des Benutzer-RAM folgendermaßen ab:



In den Systemvariablen 55 und 56 steht die Adresse. Die zweite Speicherzelle enthält die Seitenzahl.

```
POKE 56, PEEK(56) - 1
```

Am einfachsten läßt sich die Obergrenze um 256 Bytes, also um eine Seite absenken. Die Anweisungen oben lassen den Computer vom Inhalt der Zelle 56 (die die Seitenzahl enthält) 1 abziehen und den neuen Wert wieder am gleichen Platz speichern. Er reduziert also den Adreßteil für die Seitenzahl um 1. Die neue RAM-Obergrenze erhält man mit
PRINT PEEK(55) + 256* PEEK(56).

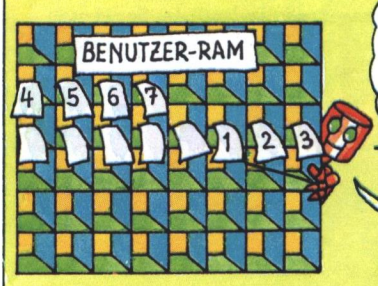
Tips für den Sinclair ZX81

Beim ZX81 speichert man Maschinencode-Programme am besten am Anfang des Benutzer-RAM. Dazu gibt man als erste Zeile des Programms Hex-Lader (siehe Seite 24) eine REM-Zeile mit so vielen Ziffern ein, wie das Maschinencode-Programm Bytes hat.

```
5 REM 1234567
```

Sieben Bytes

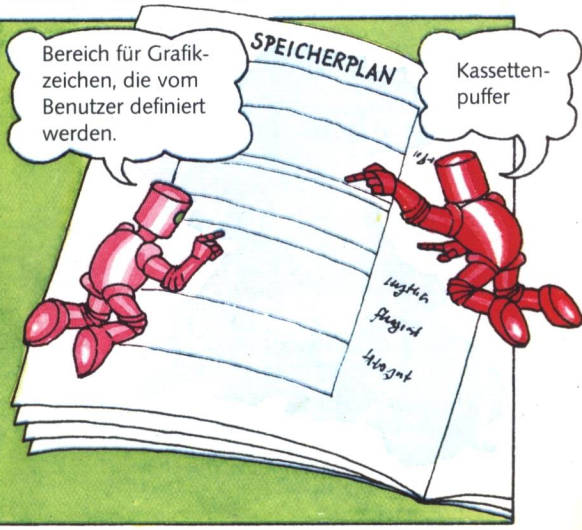
Jede Ziffer nach der REM-Anweisung belegt eine Speicherzelle. In diesen Zellen kann man nun den Maschinencode speichern. In BASIC ignoriert der Computer die Zeichen nach einer REM-Anweisung.



Zum Speichern muß man die Adresse der ersten Ziffer kennen. Das Benutzer-RAM beginnt bei 16 509. Der Computer braucht zwei Bytes für die Zeilennummer, zwei für die Zeilenlänge und eines für REM. Die erste Ziffer hat also die Adresse 16 514.

Andere Bereiche für Maschinencode

Grundsätzlich eignen sich für Maschinencode alle RAM-Bereiche, die man nicht anderweitig braucht. Will man nicht auf Kassette speichern, so kann man den Code im Kassettenpuffer unterbringen oder im Bereich für Grafikzeichen, wenn man diese nicht selbst definiert. Die Adressen dieser Bereiche findet man im Handbuch. Das Handbuch enthält vielleicht auch spezielle Tips zum Speichern von Maschinencode. Auch Zeitschriften und Bücher liefern solche Hinweise.

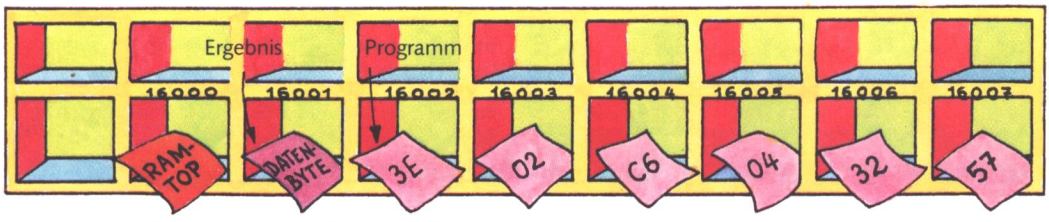


Wie man Programme lädt und zum Laufen bringt

Auf den nächsten Seiten wird gezeigt, wie man das Addierprogramm von Seite 18 in den Computer lädt und ablaufen läßt. Dazu muß man mit Hilfe der Anweisung POKE alle Bytes des Programms in die Zellen bringen, die man zur Speicherung des Maschinencodes gewählt hat. Um sich das zu erleichtern, kann man ein kurzes BASIC-Programm benutzen, den *Hex-Lader*, der auch gleich die Umrechnung von Hex-Zahlen in Dezimalzahlen für die POKE-Anweisung übernimmt. (Dieses Programm steht auf der nächsten Seite.) Zuerst muß man jedoch im Additionsprogramm die Adresse für das Ergebnis so ändern, daß sie sich für den eigenen Computer eignet. Außerdem muß man dem Programm noch eine weitere Anweisung anfügen (siehe unten).

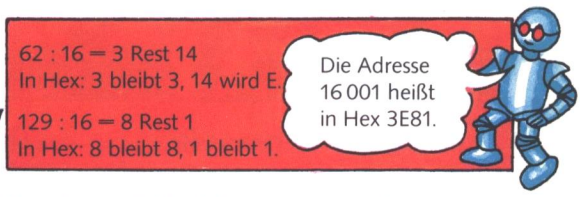
Auswahl der Adresse für das Ergebnis

Daten, die aus einem Maschinencode-Programm stammen (wie das Ergebnis des Additionsprogramms), nennt man *Datenbytes*. Es ist wichtig, die Datenbytes in Zellen zu speichern, wo sie das eigentliche Programm nicht stören. Ein guter Platz dafür ist direkt vor dem Programmbeginn.



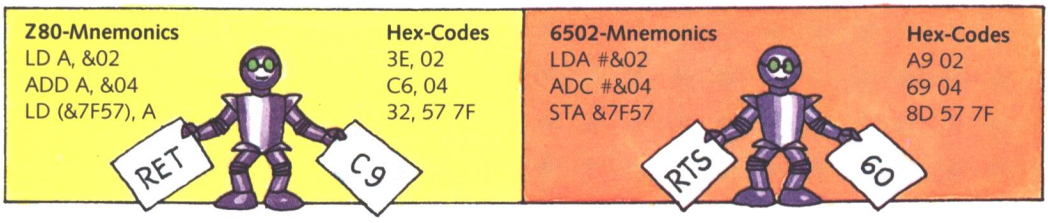
Haben Sie die RAM-Obergrenze z. B. auf 16 000 abgesenkt, dann ist 16 001 die erste Zelle für den Maschinencode. Dort sollte das Datenbyte gespeichert werden. Der Programmbeginn läge dann bei 16 002. Die Adresse für das Datenbyte muß man in eine Hex-Zahl umrechnen, bevor man sie in das Programm einschreibt.

Zur Umrechnung der Adresse in Hex teilt man die Nummer der Speicherzelle durch 256. Das Ergebnis ist die dezimale Seitenzahl, der Rest gibt die Stelle auf der Seite an (siehe Seite 11).



Diese beiden Zahlen teilt man zur Umrechnung in Hex jeweils durch 16 und ersetzt die Ergebnisse durch die entsprechenden Hex-Ziffern.

Die Anweisung „Return"



Am Ende eines Maschinencode-Programms darf die Anweisung „Return" (zurück) nicht fehlen: RET beim Z80 und RTS beim 6502. Das beendet das Maschinencode-Programm für den Computer und läßt ihn an den Ausgangspunkt zurückkehren. Sonst würde er alle folgenden Bytes als Maschinencode behandeln und bald „aussteigen". (Mehr über „Return" auf Seite 35.)

Das Programm „Hex-Lader"

Dieses Programm braucht man, um Hex-Codes in den Computer zu laden. Wenn man den Lader benutzen will, schreibt man die Hex-Codes des Maschinencode-Programms in Zeile 160, gefolgt von END (als Zeichen für den Computer, daß keine Codes mehr kommen). Bei Zeile 40 liest der Computer die Ziffernpaare, rechnet in den Zeilen 70 bis 110 die Dezimalwerte aus und lädt sie bei Zeile 130 in den Speicher*.

```
10 PRINT "STARTADRESSE
   ZUM SPEICHERN DES CODES?"
20 INPUT A
30 LET C=0
40 READ H$
50 IF H$="END" THEN GOTO 180
60 IF LEN(H$)<>2 THEN GOTO 170
70 LET X=(ASC(H$)-48)*16
80 IF ASC(H$)>57 THEN LET X=
   (ASC(H$)-55)*16
90 LET Y=ASC(RIGHT$(H$,1))
100 IF Y<58 THEN LET X=X+Y-48
110 IF Y>57 THEN LET X=X+Y-55
120 IF X<0 OR X>255 THEN GOTO 170
130 POKE A+C,X
140 LET C=C+1
150 GOTO 40
155 REM DATEN SIND NUR BEISPIELE
160 DATA EF,F6,E2,A9,END
170 PRINT "BAD DATA"
180 STOP
```

A ist die Anfangsadresse des Bereichs, in dem der Code gespeichert werden soll.

C ist ein Zähler.

Gibt H$ den Wert eines Ziffernpaars aus Zeile 160.

Prüft, ob durch END der Schluß der Codeeingaben angezeigt wird.

Prüft, ob H$ zwei Ziffern enthält. Wenn nicht, Sprung zu Zeile 170.

Gibt X den Dezimalwert der ersten Ziffer.

Gibt Y den Dezimalwert der zweiten Ziffer und addiert Y zu X.

Prüft, ob X zwischen 0 und 255 liegt. Wenn nicht, Sprung zu Zeile 170.

Beim erstenmal ist C = 0; also kommt X in die Speicherzelle A.

Addiert 1 zu C, damit X beim nächsten Durchgang in die nächste Speicherzelle (A + 1) kommt.

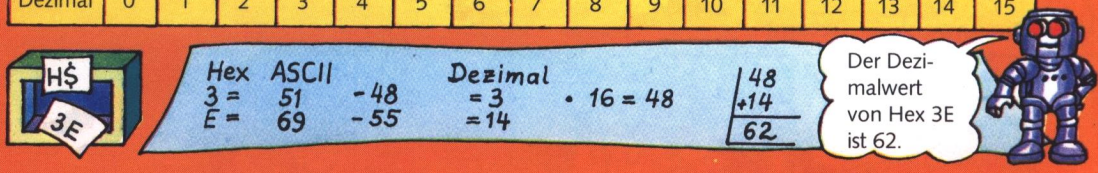
Springt zurück zum Lesen des nächsten Hex-Codes.

Setzen Sie hier die Hex-Codes ein, gefolgt von END.

Zeigt BAD DATA („schlechte Daten") auf dem Bildschirm an, wenn solche in Zeile 60 oder 120 gefunden wurden. Danach STOP.

So funktioniert der Hex-Lader

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
ASCII	48	49	50	51	52	53	54	55	56	57	65	66	67	68	69	70
	minus 48										minus 55					
Dezimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



In den Zeilen 70 und 80 wird die erste Ziffer von H$ in eine Dezimalzahl umgewandelt und in X gespeichert. Das BASIC-Wort ASC macht aus einem Zeichen die zugehörige ASCII-Codezahl. Je nach Größe wird 48 oder 55 abgezogen (siehe Tabelle oben). Das Ergebnis mit 16 multipliziert, ergibt den gesuchten Dezimalwert.

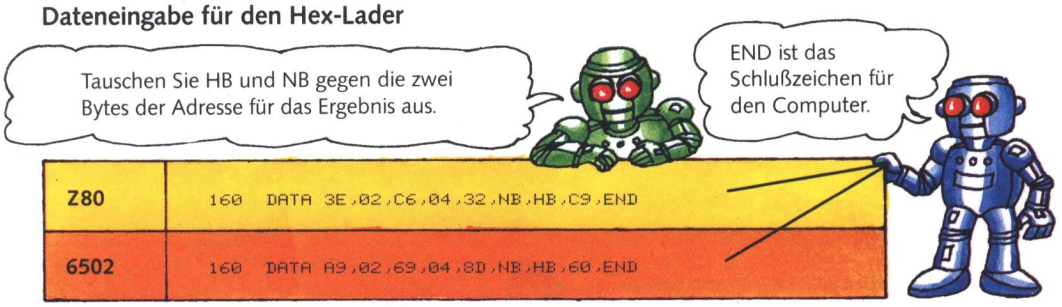
In den Zeilen 90 bis 120 geschieht das gleiche mit der zweiten Ziffer von H$, und das Ergebnis wird zu X addiert. Hier multipliziert man allerdings nicht mit 16, weil die zweite Ziffer die Anzahl der Einer in der Hex-Zahl angibt und nicht, wie zuvor, die Anzahl der 16er. So erhält X den Dezimalwert der zwei Hex-Ziffern.

* Beim Sinclair Spectrum schreibt man CODE statt ASC und setzt die Ziffernpaare des Hex-Codes in Anführungszeichen. Änderungen für den Sinclair ZX81 und Atari-Computer stehen auf Seite 44.

So benutzt man den Hex-Lader

Nun kann man den Hex-Lader benutzen, um das Addierprogramm auszuprobieren. Das Programm ist nicht besonders aufregend, aber es zeigt, wie Maschinencode funktioniert. Geben Sie den Hex-Lader in Ihren Computer ein. Dabei tauschen Sie die Beispieldaten in Zeile 160 gegen die Codes des Addierprogramms (siehe unten) aus.

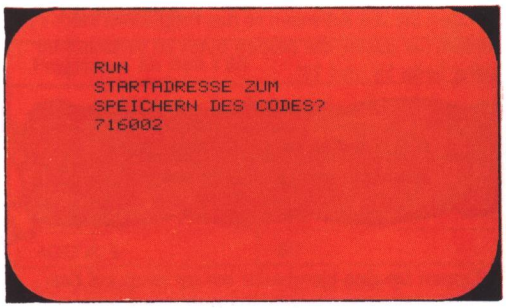
Dateneingabe für den Hex-Lader

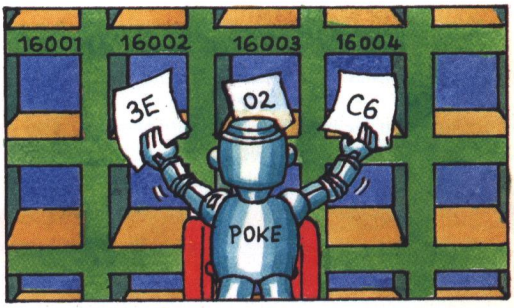


Z80	160 DATA 3E,02,C6,04,32,NB,HB,C9,END
6502	160 DATA A9,02,69,04,8D,NB,HB,60,END

Dies sind die Hex-Codes für das Addierprogramm. Man muß die Buchstaben NB (niederwertiges Byte) und HB (höherwertiges Byte) durch die zwei Bytes der Adresse ersetzen, unter der das Ergebnis gespeichert werden soll. Zuerst kommt die Stelle auf der Seite, dann die Seitenzahl.

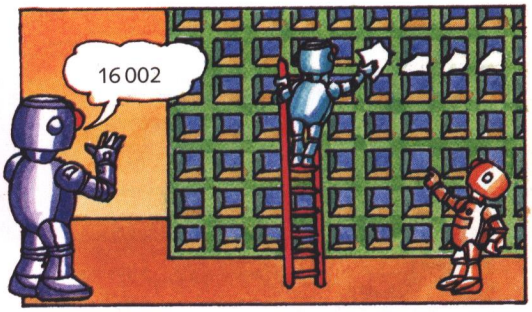
Der Hex-Lader wird gestartet





Durch RUN wird der Hex-Lader gestartet. Wenn er nach der Adresse fragt, dann gibt man die erste Adresse nach der Speicherzelle für das Ergebnis ein, und zwar als Dezimalzahl, wie sie für die POKE-Anweisung gebraucht wird.

Das Maschinencode-Programm wird gestartet



Die Anweisung zum Starten von Maschinencode-Programmen ist von Gerät zu Gerät verschieden. Sie kann CALL, PRINT USR oder SYS lauten, gefolgt von der ersten Adresse des Maschinencodes. Schlagen Sie das im Handbuch nach. Wenn der Computer diese Anweisung erhält, dann springt er zu der angegebenen Adresse und beginnt mit der Ausführung der Maschinencode-Anweisungen.

Das Ergebnis wird angezeigt

```
PRINT PEEK(16001)
```

Der Computer führt die Maschinencode-Anweisungen aus und speichert das Ergebnis unter der angegebenen Adresse.

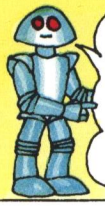
```
PRINT PEEK(16001)
6
```

Wenn man das Ergebnis sehen will, verwendet man PRINT PEEK mit der Adresse des Ergebnisses. Es wird dezimal angezeigt.

Übungsprogramme

Mit den Grundkenntnissen, die Sie jetzt erworben haben, können Sie schon selbst kleine Programme schreiben. Die Checkliste unten hilft Ihnen, an alles zu denken, was man dabei beachten muß.
(Die Lösungen finden Sie auf Seite 46.)

1. Ändern Sie das Addierprogramm, so daß es 25 und 73 (dezimal) addiert und das Ergebnis speichert.
2. Versuchen Sie, ein Programm zu schreiben, das 64, 12 und 14 (dezimal) addiert und das Ergebnis speichert.



Das Addierprogramm kann nur Summen bilden, die 255 nicht überschreiten. Auf Seite 28 steht, wie man größere Summen bildet.

Checkliste für Maschinencode

1. Schreiben Sie Ihr Programm in Assemblersprache und die Daten in Hex.

2. Schlagen Sie die Hex-Codes für die Mnemonics in der Liste am Ende des Buches nach.

Vergessen Sie nicht das END nach den Hex-Codes im Hex-Lader!

3. Setzen Sie an den Schluß des Programms die Return-Anweisung (siehe Seite 23).

4. Zählen Sie die Bytes und reservieren Sie dafür Platz im RAM (siehe Seite 20 bis 22).

Notieren Sie sich die Adressen der Datenbytes und die Anfangsadresse des Programms!

5. Bestimmen Sie die Hex-Werte der Adressen, die Sie für Datenbytes brauchen (siehe Seite 23).

6. Fügen Sie die Adressen in das Programm ein – denken Sie daran, die beiden Bytes zu vertauschen (siehe Seite 18 und 19).

Prüfen Sie sorgfältig die Hex-Codes, bevor Sie den Hex-Lader starten!

7. Tippen Sie den Hex-Lader ein (er kann auf Kassette gespeichert werden), mit den Hex-Codes und END in Zeile 160 (siehe Seite 24).

Sollte Ihr Programm nicht laufen, so überprüfen Sie, ob Sie die richtigen Hex-Codes benutzt haben.

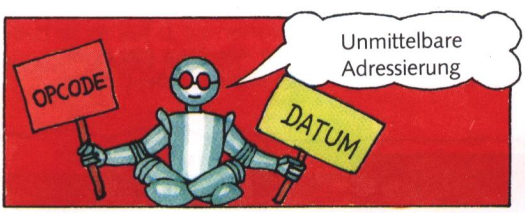
8. Starten Sie den Hex-Lader und geben Sie die erste Adresse für den Maschinencode ein (siehe Seite 25).

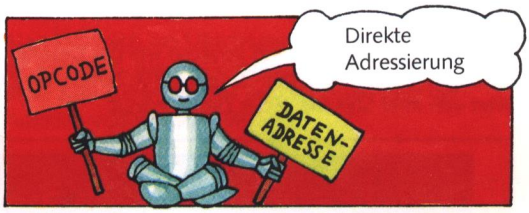
9. Starten Sie das Maschinencode-Programm mit der entsprechenden Anweisung für Ihren Computer, gefolgt von der ersten Adresse des Maschinencodes als Dezimalzahl (siehe Seite 25).

Nach jeder Änderung des Programms müssen Sie den Hex-Lader erneut laufen lassen, damit er die neuen Bytes (mit POKE) in den Speicher bringt.

Wie man Daten im Speicher findet

Im vorhergehenden Programm waren die Daten im Programm selbst enthalten. In einem solchen Fall spricht man von *unmittelbarer Adressierung.* Manchmal soll der Computer jedoch Daten bearbeiten, die irgendwo im Speicher stehen. In diesem Fall muß der Operand einer Anweisung die Adresse sein, unter der der Computer die Daten findet. Das nennt man *direkte* oder *absolute Adressierung.*





Dies sind nur zwei von mehreren Verfahren, mit denen man dem Computer mitteilt, wo er die Daten findet, mit denen er arbeiten soll. Die Verfahren heißen *Adressierungsarten.* Die Werte des Hex-Codes für eine Anweisung richten sich nach der Adressierungsart, die jeweils angewendet wird.

Addierprogramm für Zahlen aus dem Speicher

Hier ist ein Programm, mit dem Zahlen aus dem Speicher geholt und addiert werden. Vergleichen Sie die Hex-Codes in diesem Programm (direkte Adressierung) mit denen aus dem vorhergehenden Programm (unmittelbare Adressierung).

Programm für den Z80		
Mnemonics	**Hex-Codes**	**Bedeutung**
LD A, (Adresse 1)	3A, Adresse 1	Lädt die Zahl aus Adresse 1 in den Akkumulator.
LD B, A	47	Lädt die Zahl im Akkumulator in Register B.
LD A, (Adresse 2)	3A, Adresse 2	Lädt die Zahl aus Adresse 2 in den Akkumulator.
ADD A, B	80	Addiert die Zahl in Register B zum Akkumulator.
LD (Adresse 3), A	32, Adresse 3	Speichert den Inhalt des Akkumulators in Adresse 3.
RET	C9	Return

Um zwei Zahlen aus dem Speicher zu addieren, muß man sie zuerst in die Register holen. Man kann dafür die Register A (Akkumulator) und B nehmen. Da sich außer dem Akkumulator kein Register direkt aus dem Speicher laden läßt, muß man die erste Zahl in Register A laden und dann in Register B übertragen.

Programm für den 6502		
Mnemonics	**Hex-Codes**	**Bedeutung**
LDA Adresse 1	AD Adresse 1	Lädt die Zahl aus Adresse 1 in den Akkumulator.
ADC Adresse 2	6D Adresse 2	Addiert die Zahl aus Adresse 2 zum Akkumulator.
STA Adresse 3	8D Adresse 3	Speichert den Inhalt des Akkumulators in Adresse 3.
RTS	60	Return

So läuft das Programm

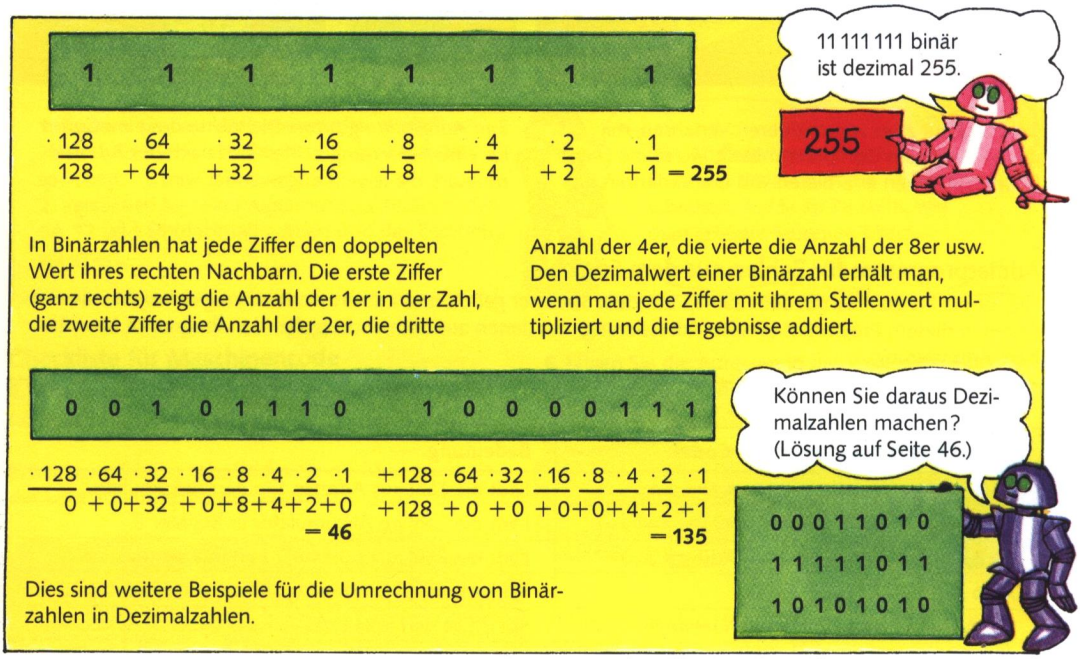
Gehen Sie mit diesem Programm die Schritte aus der Checkliste auf der vorigen Seite durch, um es zum Laufen zu bringen. Davor müssen Sie jedoch die Zahlen, die addiert werden sollen, mit POKE in den Speicher bringen. Reservieren Sie dazu am besten Zellen vor dem Maschinencode-Programm, damit die Datenbytes getrennt von den Anweisungen bleiben. Diese Adressen und eine dritte für das Ergebnis schreiben Sie als Hex-Zahlen in das Programm. PRINT PEEK (Adresse 3) zeigt das Ergebnis.

Wie man mit großen Zahlen rechnet

Die Programme auf den vorigen Seiten verarbeiten nur Zahlen, deren Summe nicht größer ist als 255. Das ist die größte Zahl, die sich mit den acht Bits in einem Register oder einer Speicherzelle darstellen läßt. Für größere Zahlen muß man etwas mehr über das binäre Zahlensystem und das Übertragskennzeichen (Carry Flag) wissen. Auf Seite 30 finden Sie ein Addierprogramm für große Zahlen.

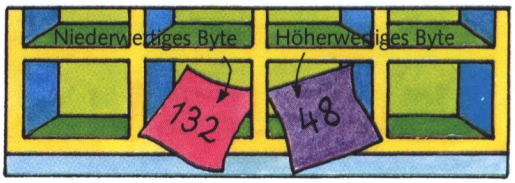
Das Binärsystem

Das Binärsystem funktioniert genauso wie das Dezimal- und das Hexadezimalsystem. Allerdings gibt es im Binärsystem nur zwei Ziffern, 0 und 1. Für Zahlen, die größer sind als 1, nimmt man mehrere Ziffern, deren Wert von ihrer Stellung in der Zahl abhängt.



So übergibt man dem Computer große Zahlen

Im Computer werden Zahlen, die größer sind als 255, in zwei Bytes gespeichert, einem höherwertigen und einem niederwertigen, genauso wie Adressen. Das höherwertige Byte zeigt die Anzahl der 256er, das niederwertige den Rest. Wie bei Adressen erwartet der Computer das niederwertige Byte vor dem höherwertigen. Man muß sie also in dieser Reihenfolge im Speicher ablegen.



Um dem Computer Zahlen zu übergeben, die größer sind als 255, muß man die Werte beider Bytes ausrechnen. Dafür teilt man die Zahl durch 256. Das Ergebnis ist der Dezimalwert des höherwertigen Bytes, der Rest der des niederwertigen Bytes.

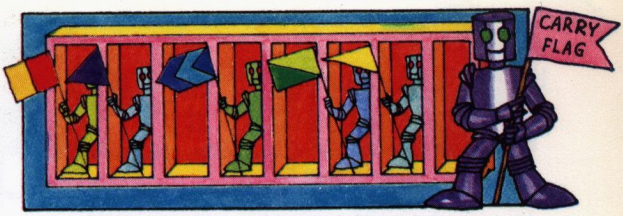
Um eine solche Zahl in einem Maschinencode-Programm zu verwenden, muß man die Bytes in Hex umwandeln. Das macht man, wie es auf Seite 11 beschrieben ist: Man teilt die Bytes durch 16 und wandelt die Ergebnisse und Reste in Hex-Ziffern um.

307; 21 214; 759; 1 023.

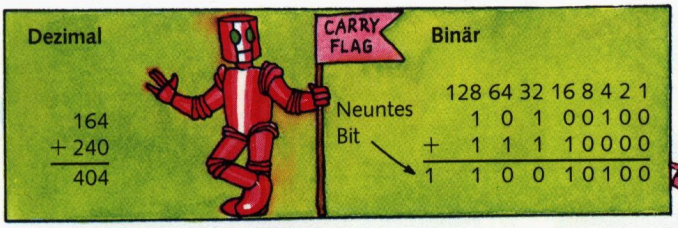
Berechnen Sie die Dezimalwerte des höher- und niederwertigen Bytes für diese Zahlen. Wie lauten sie in Hex? (Lösung auf Seite 46.)

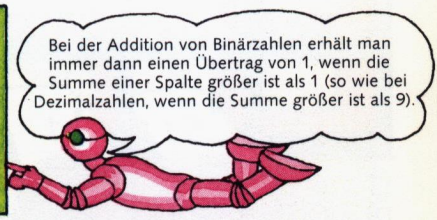
Das Übertragskennzeichen

Das Übertragskennzeichen oder Carry Flag ist ein Bit im Flagregister bzw. Prozessor-Statusregister, das anzeigt, wann ein Rechenergebnis größer als 255 ist und nicht in ein Byte (acht Bits) paßt. Ist das der Fall, dann setzt der Computer das Übertragskennzeichen auf 1. Man sagt: „Er setzt (englisch set) das Flag." Setzt er es auf 0, so nennt man das „zurücksetzen" (englisch clear).



Man kann sich das Übertragskennzeichen als neuntes Bit vorstellen, das den Übertrag aus der achten Stelle einer Zahl aufnimmt. Das Beispiel unten zeigt die Addition von 164 und 240.

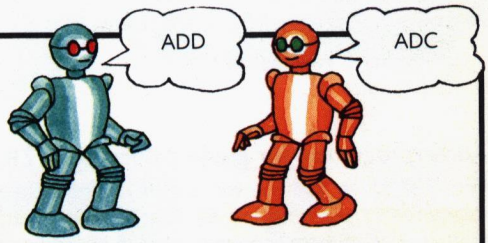




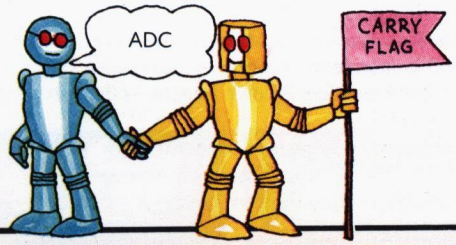
Die Summe ist 404. Dafür braucht man binär neun Bits. Das neunte Bit zeigt die Anzahl der 256er in der Zahl. Im Computer ist dieses Bit das Übertragskennzeichen.

Der Übertrag beim Z80

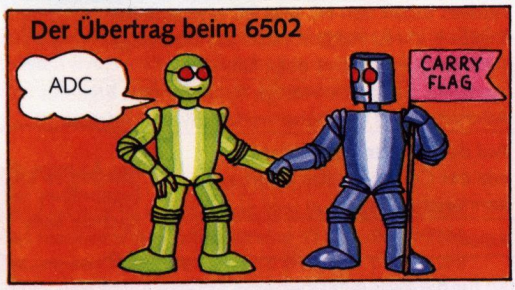
Der Z80 hat zwei Addieranweisungen: ADD und ADC. Mit ADD addiert der Computer zwei Zahlen, ohne einen Übertrag aus früheren Rechnungen zu beachten. Aber er setzt das Übertragskennzeichen auf 1 oder 0, je nachdem ob jetzt ein Übertrag vorhanden ist oder nicht.



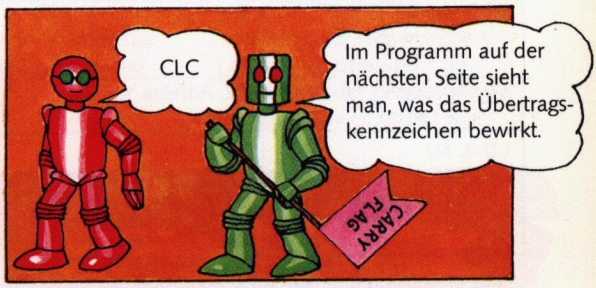
ADC ist das Mnemonic für englisch **ad**d with **c**arry (= addiere mit Übertrag): Der Computer addiert zu zwei Zahlen auch noch das Übertragskennzeichen und setzt es dann, je nach dem Ergebnis, auf 0 oder 1. Wenn man mehrere Additionen auszuführen hat, benutzt man bei der ersten am besten die Anweisung ADD, damit ein Übertrag aus vorangegangenen Rechnungen das Ergebnis nicht verfälscht. Danach benutzt man ADC für den Fall, daß ein Übertrag vorhanden ist.



Der Übertrag beim 6502



Der 6502 hat nur die Addieranweisung ADC, er berücksichtigt also stets den Inhalt des Übertragskennzeichens. Daher muß man am Beginn von



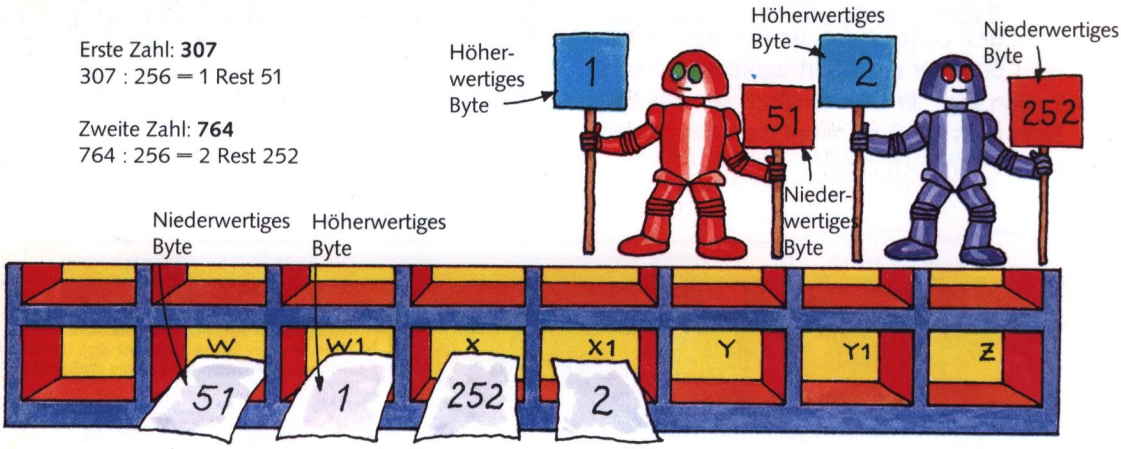
Additionen das Übertragskennzeichen mit CLC (Mnemonic für englisch **cl**ear **c**arry = Übertrag löschen) zurücksetzen.

Programme für große Zahlen

Für die Programme auf diesen beiden Seiten muß man die Zahlen, die man addieren will, in die höherwertigen und niederwertigen Bytes umwandeln und durch POKE in den Speicher bringen (im Beispiel unten die Zahlen 307 und 764).

Erste Zahl: **307**
307 : 256 = 1 Rest 51

Zweite Zahl: **764**
764 : 256 = 2 Rest 252



Diese Bytes muß man durch POKE in die Speicherzellen am Anfang des Bereichs bringen, den man für den Maschinencode reserviert hat. (Das niederwertige Byte jeder Zahl kommt zuerst.) Im Bild oben sind die zwei Bytes der ersten Zahl in den Zellen W und W1 gespeichert und die Bytes für die zweite Zahl in X und X1. Für das Ergebnis braucht man drei Zellen: eine für das niederwertige Byte, eine für das höherwertige und eine für einen möglichen Übertrag.

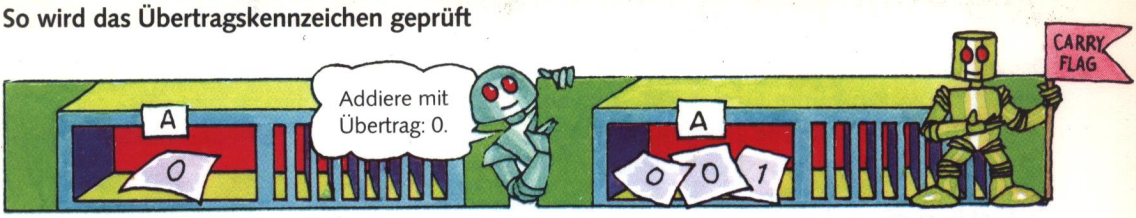
Addierprogramm für große Zahlen beim Z80

Beim Z80 ist die Addition von großen Zahlen recht einfach, weil man für die zwei Bytes einer großen Zahl ein Registerpaar benutzen kann. Ein Paar können z. B. die Register H und L sein, ein anderes die Register B und C. Statt im Akkumulator kann man auch in HL addieren. Unten sind die Mnemonics und die Hex-Codes für das Programm angegeben. Ein Blick auf die Speichereinteilung hilft Ihnen vielleicht, das Programm zu verstehen.

Mnemonics	Hex-Codes	Bedeutung
LD HL, (Adresse W)	2A, Adresse W	Lädt Byte von Adresse W (niederwertiges Byte) in Register L und Byte von Adresse W1 (höherwertiges Byte) in Register H (erste Zahl).
LD BC, (Adresse X)	ED 4B, Adresse X Dieser Opcode ist zwei Bytes lang.	Lädt Byte von Adresse X (niederwertiges Byte) in Register C und Byte von Adresse X1 (höherwertiges Byte) in Register B (zweite Zahl).
ADD HL, BC	09	Addiert Inhalt von HL und BC und lädt das Ergebnis in HL. Addiert nicht das Übertragskennzeichen, setzt es aber, falls nötig.
LD (Adresse Y), HL	22, Adresse Y	Speichert das niederwertige Byte aus dem Ergebnis in Adresse Y und das höherwertige Byte in Adresse Y1
LD A, &0 ADC A, &0 LD (Adresse Z), A	3E, 0 CE, 0 32, Adresse Z	Auf der nächsten Seite steht, wie der Computer das Übertragskennzeichen prüft.
RET	C9	Return

In das Programm einfügen muß man noch die Hex-Werte der Adressen W, X, Y und Z (richtige Reihenfolge beachten!). Benutzt man Registerpaare, so muß man für die zwei Bytes nur eine Adresse angeben. Der Computer ordnet dem zweiten Register automatisch die nächstfolgende Adresse zu.

So wird das Übertragskennzeichen geprüft

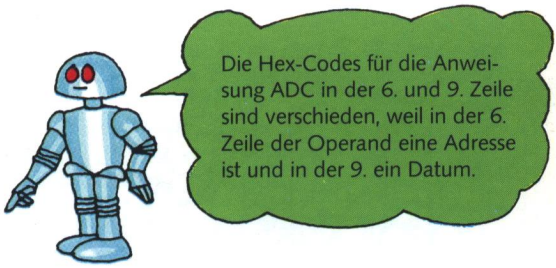


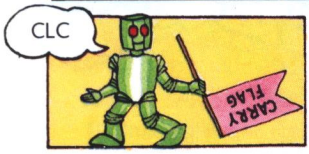
In der 5. bis 7. Zeile des Z80-Programms wird der Wert des Übertragskennzeichens geprüft und in den Speicher gebracht. Dazu wird zuerst der Akkumulator mit 0 geladen (5. Zeile). Dann werden 0 und das Übertragskennzeichen addiert (ADC in der 6. Zeile). Ist das Übertragskennzeichen bei der vorhergehenden Addition gesetzt worden, so steht jetzt die 1 aus dem Übertrag im Akkumulator (sonst 0). Dieser Wert wird in der 7. Zeile unter der Adresse Z gespeichert.

Addierprogramm für große Zahlen beim 6502

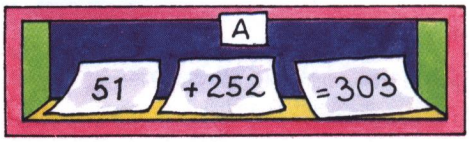
Mit diesem Programm für den 6502 kann man Zahlen addieren, die größer sind als 255. Dazu muß man die höherwertigen und niederwertigen Bytes der beiden Zahlen bestimmen und, wie auf der vorigen Seite beschrieben, mit POKE in den Speicher bringen.

Mnemonics	Hex-Codes
CLC	18
LDA Adresse W	AD Adresse W
ADC Adresse X	6D Adresse X
STA Adresse Y	8D Adresse Y
LDA Adresse W1	AD Adresse W1
ADC Adresse X1	6D Adresse X1
STA Adresse Y1	8D Adresse Y1
LDA #&0	A9 00
ADC #&0	69 00
STA Adresse Z	8D Adresse Z
RTS	60

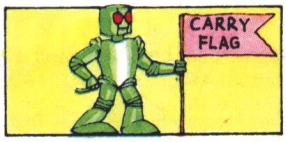




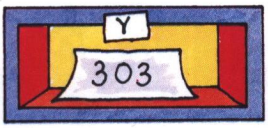
Der erste Programmschritt macht das Übertragskennzeichen zu 0, falls es vorher gesetzt wurde.

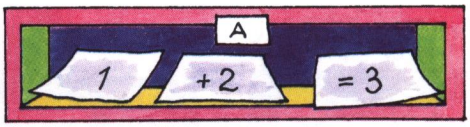


Dann wird das niederwertige Byte der ersten Zahl in den Akkumulator geholt und dazu das erste Byte der zweiten Zahl addiert (2. und 3. Zeile).

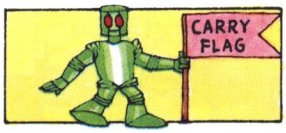


Ist das Ergebnis größer als 255, so wird das Übertragskennzeichen gesetzt.





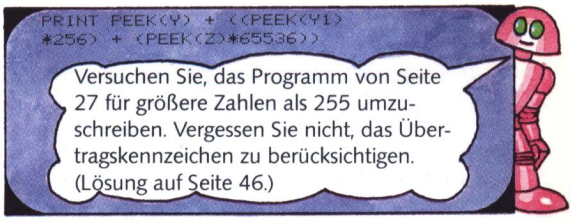
Das Ergebnis wird unter Adresse Y gespeichert (4. Zeile). Dann werden die zweiten Bytes und der Übertrag von vorher (falls vorhanden) addiert. Das Ergebnis wird unter Adresse Y1 gespeichert (7. Zeile).



Die 8. bis 10. Zeile verarbeiten das Übertragskennzeichen genauso, wie es oben auf dieser Seite gezeigt wird.

Das Ergebnis

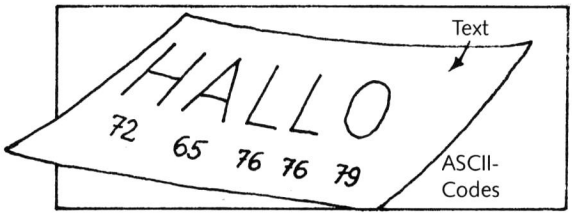
Das Ergebnis ist in Form von drei Bytes gespeichert. Das niederwertige Byte (Adresse Y) gibt die Anzahl der 1er, das höherwertige Byte (Adresse Y1) die der 256er und der Übertrag (Adresse Z) die Anzahl der 65 536er. Die Anweisungen rechts zeigen das Ergebnis. (Y, Y1 und Z müssen Sie durch die passenden Adressen für Ihren Computer ersetzen.)

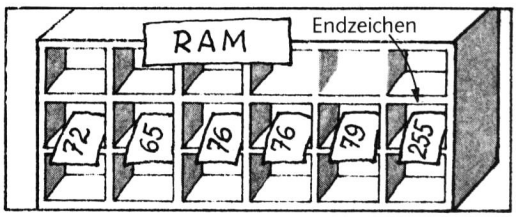


Wie man Text auf den Bildschirm bringt

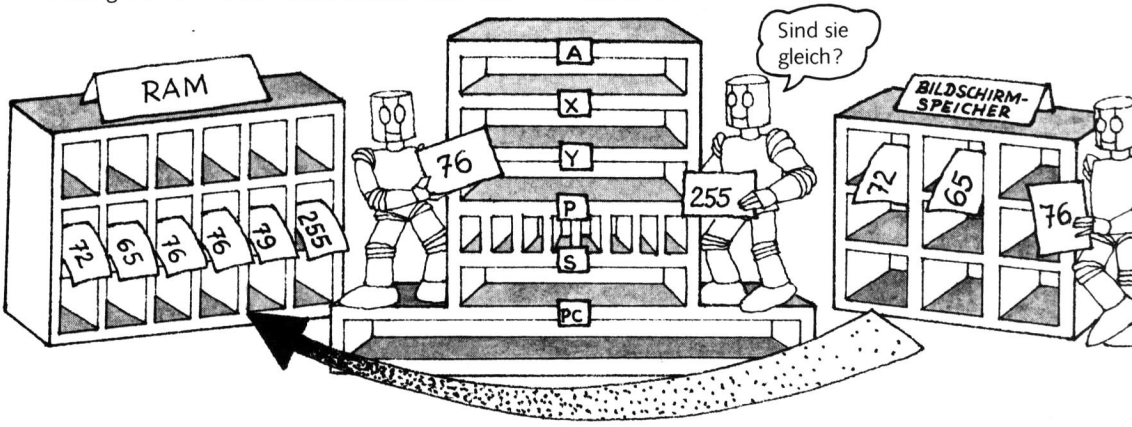
Das nächste Programm zeigt, wie man mit Maschinencode einen Text auf den Bildschirm bringen kann. Das Programm für den Z80 steht auf Seite 33, das Programm für den 6502 auf Seite 34. Beide Programme arbeiten grundsätzlich nach der gleichen Methode, unterscheiden sich aber etwas wegen der unterschiedlichen Mikroprozessoren*.

So funktioniert das Programm



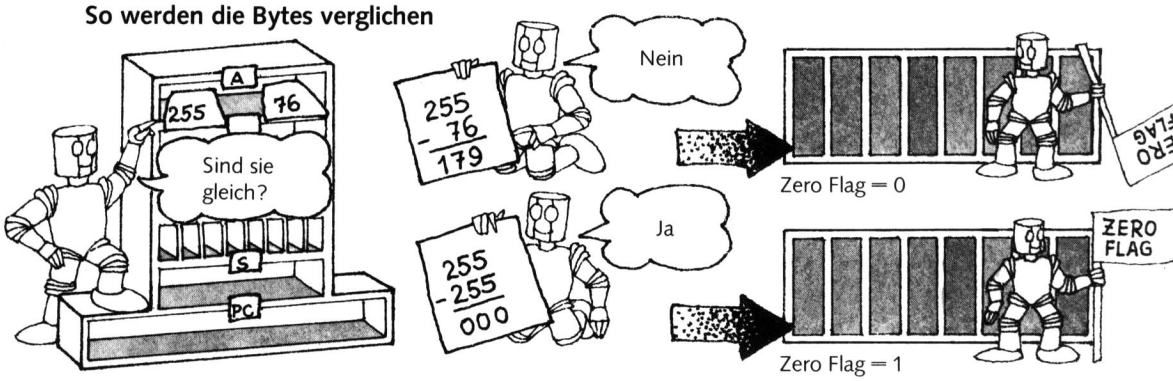


Zuerst speichert man mit POKE den Zeichencode für jeden Buchstaben des Textes in einer Zelle am Anfang des freien RAM-Bereichs. Jeder Buchstabe braucht ein Byte. Ans Ende des Textes setzt man den Code 255, damit der Computer weiß, daß hier der Text endet.



Das Programm holt jedes Textbyte in den Akkumulator und vergleicht es mit 255. Wenn das Textbyte nicht 255 ist, dann kommt es in den Bildschirmspeicher und wird automatisch angezeigt. Dann springt der Computer an den Anfang des Programms und holt das nächste Textbyte aus dem Speicher.

So werden die Bytes verglichen



Mit dem Opcode CP beim Z80 und CMP beim 6502 weist man den Computer an, ein Byte mit dem Inhalt des Akkumulators zu vergleichen. Dazu zieht der Computer ein Byte vom anderen ab. (Nur zur Prüfung, die Bytes bleiben unverändert.) Bei gleichen Bytes ist das Ergebnis 0, und das Zero Flag wird auf 1 gesetzt. Sind die Bytes nicht gleich, so wird das Zero Flag auf 0 gesetzt. Je nach dem Wert des Zero Flags kann man nun zu einer anderen Programmstelle springen oder mit der nächsten Anweisung fortfahren.

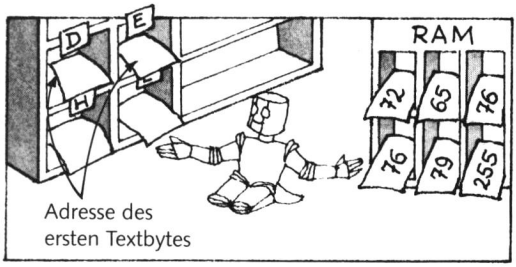
* Beim Sinclair Spectrum erhalten Sie keinen lesbaren Text auf dem Bildschirm, weil der Bildschirmspeicher anders aufgebaut ist.

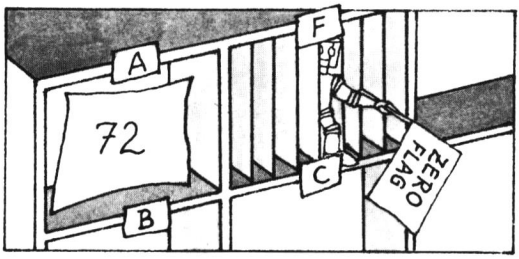
Textprogramm für den Z80

Hier folgen die Mnemonics und Hex-Codes für den Z80. Zuerst bringt man mit POKE die Textbytes in freie Speicherzellen. Dann werden in der 1. und 2. Zeile die Adressen eingefügt. Die letzte Anweisung im Programm läßt den Computer zur 3. Zeile springen. Nach dem Opcode für JP(C3) muß man die Adresse einfügen, unter der im eigenen Computer die Anweisung der 3. Zeile (LD A) gespeichert ist.

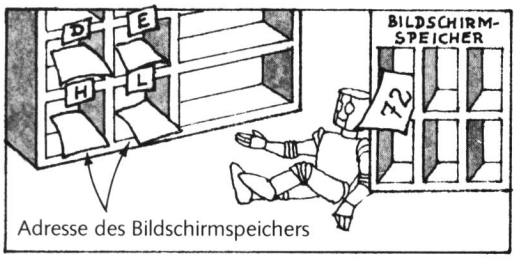
Mnemonics	Hex-Codes
LD HL, Bildschirmspeicher-Adr.	21, Bildschirmspeicher-Adr.
LD DE, Textadresse	11, Textadresse
LD A, (DE)	1A
CP, &FF	FE, FF
RET Z	C8
LD (HL), A	77
INC, DE	13
INC, HL	23
JP, Adresse der 3. Anweisung	C3, Adresse der 3. Anweisung

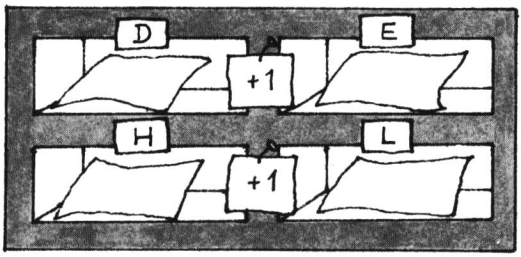
In diesem Programm werden die Registerpaare HL und DE als Zeiger benutzt, die auf die Adresse deuten, von oder zu denen der Computer Daten bringen soll. Das nennt man *indirekte Adressierung*. Diese Adressierungsart verwenden die Anweisungen in der 3. und 6. Zeile.
In den ersten beiden Zeilen lädt der Computer die Bildschirmspeicher-Adresse (unter der er Daten speichern soll) in das Registerpaar HL und die Textadresse (von der er Daten abholt) in das Registerpaar DE.





LD A, (DE) sagt dem Computer, daß er die Adresse in DE lesen und von dieser Adresse ein Byte in den Akkumulator holen soll. Das ist indirekte Adressierung. Dann vergleicht er das Byte im Akkumulator mit &FF (Hex für 255). RET Z weist den Computer an, zum BASIC zurückzukehren, wenn das Zero Flag 1 ist (wenn also das Byte gleich 255 war). Ist das Zero Flag 0, so fährt er mit der nächsten Anweisung fort.





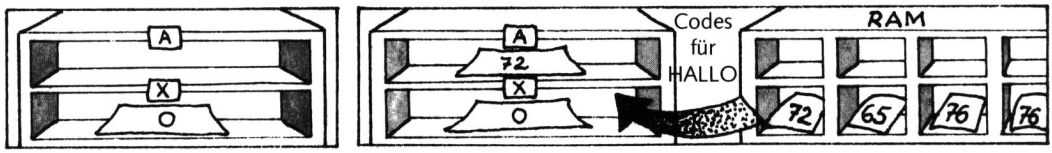
LD (HL), A verwendet auch indirekte Adressierung. Der Computer liest die Adresse in HL und speichert in der Zelle mit der Adresse den Inhalt des Akkumulators (das Textbyte). INC ist das Mnemonic für englisch **inc**rement, das bedeutet „erhöhen". In der 7. und 8. Zeile erhöht der Computer die Adresse in DE und HL um 1, so daß er nach dem Sprung zur 3. Zeile das Textbyte von der nächsten Adresse holt und in der nächsten Bildschirmspeicher-Adresse ablegt (6. Zeile).

Textprogramm für den 6502

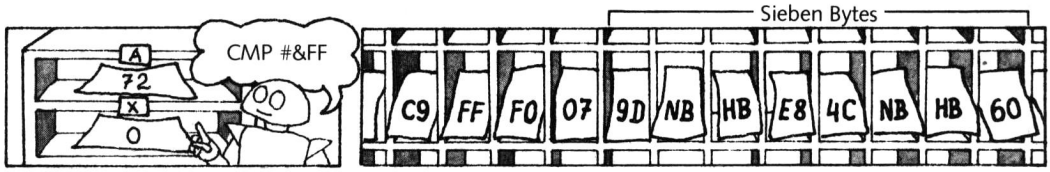
Hier folgen die Mnemonics und Hex-Codes für den 6502. Zuerst bringt man mit POKE die Textbytes in freie Speicherzellen, gefolgt von 255, dem Endzeichen. Dann fügt man in die 2. Programmzeile den Hex-Wert der Adresse mit dem ersten Textbyte ein und in die 5. Zeile die Adresse des Bildspeichers.
In die 7. Zeile kommt nach der Sprunganweisung (JMP von englisch **jump** = springe) die Adresse, unter der die zweite Anweisung des Programms im Computer gespeichert ist. Dorthin springt der Computer zurück, um das Programm mit neuen Adressen für Textbytes und Bildschirmspeicher zu wiederholen.

Mnemonics	Hex-Codes
LDX #&00	A2 00
LDA Textadresse, X	BD Textadressse
CMP #&FF	C9 FF
BEQ zur Anweisung RTS	F0 07
STA Bildschirmspeicher-Adr., X	9D Bildschirmspeicher-Adr.
INX	E8
JMP Adresse der 2. Anweisung	4C Adresse der 2. Anweisung
RTS	60

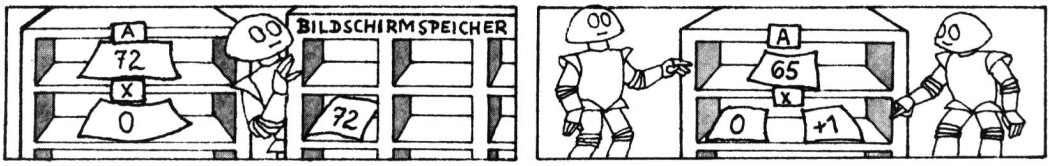
Dieses Programm verwendet eine weitere Adressierungsart, die *indizierte Adressierung*. Der Inhalt des Registers X (oder Y) wird zum Operanden addiert. Das ergibt die Adresse für die Daten (2. und 5. Zeile).



In der 1. Zeile wird 0 im Register X abgelegt. Die zweite Anweisung verwendet indizierte Adressierung. Der Computer addiert den Inhalt des Registers X zur Adresse in der Anweisung. Das Ergebnis ist die Adresse der Daten, die in den Akkumulator geladen werden sollen (ein Textbyte).



CMP (von englisch **comp**are = vergleiche) in der 3. Zeile vergleicht das Byte im Akkumulator mit FF (Hex für 255), dem Endzeichen. Sind die Bytes gleich, wird das Zero Flag auf 1 gesetzt. Der nächsten Anweisung, BEQ (von englisch **b**ranch if **eq**ual = verzweige, wenn gleich), folgt eine Zahl (in Hex hier 07), die dem Computer angibt, wie viele Stellen er weiterspringen muß. Wenn das Textbyte gleich 255 ist, soll der Computer zur Anweisung RTS verzweigen; bis dorthin sind es sieben Bytes.



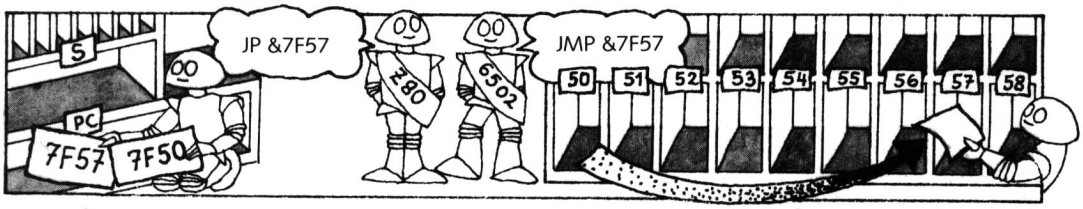
In der 5. Zeile wird schließlich der Inhalt des Akkumulators mit indizierter Adressierung (Adresse in der Anweisung plus Inhalt des Registers X) in den Bildschirmspeicher gebracht.
INX bedeutet **in**crement **X** (erhöhe X): Der Computer addiert 1 zum Inhalt des Registers X, damit wird X gleich 1. Nach dem Sprung zur zweiten Anweisung holt der Computer also das nächste Textbyte und speichert es in der nächsten Zelle des Bildschirmspeichers.

Sprünge und Verzweigungen

Wenn man den Computer an einer anderen Stelle des Programms fortfahren lassen will, so nennt man das *Verzweigung*. Davon gibt es drei Arten: *Sprung*, *Unterprogramm* und *bedingte Verzweigung*. Ein Sprung schickt den Computer einfach zu einer neuen Adresse. Eine bedingte Verzweigung macht der Computer nur, wenn eine bestimmte Bedingung erfüllt ist. Darüber finden Sie mehr auf der nächsten Seite.

Der Programmzähler

Der Programmzähler ist ein spezielles 16-Bit-Register, das gewöhnlich die Adresse des nächsten Bytes enthält, das der Computer aus dem Speicher holen soll. Der Computer „liest" die Adresse und holt das angegebene Byte dort ab. Dann erhöht er die Zahl im Programmzähler um 1, so daß dieser die Adresse der nächsten Speicherzelle enthält.

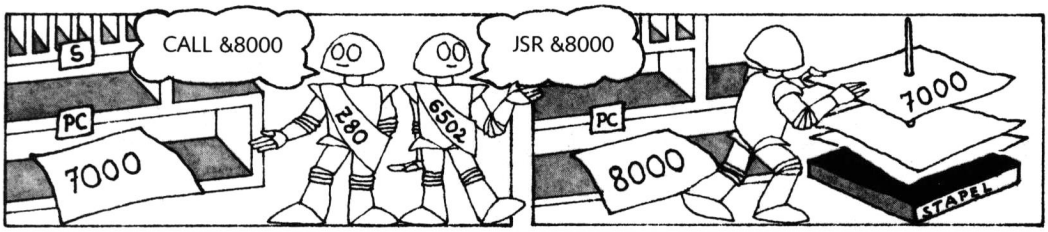


Sprung

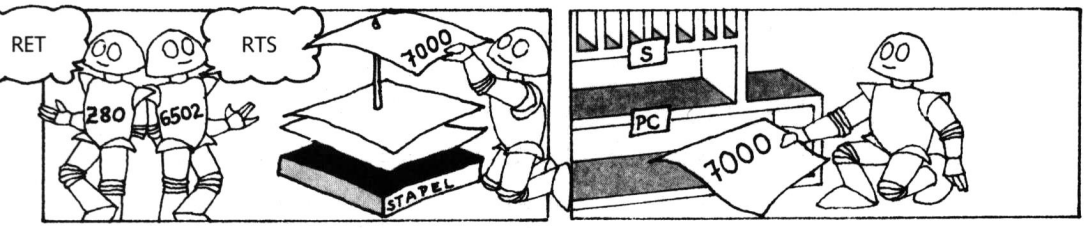
Wenn der Computer zu einer bestimmten Adresse springen oder verzweigen soll, so wird diese Adresse in den Programmzähler geladen, und der Computer arbeitet von dort aus weiter. Die Mnemonics für einen Sprung beim Z80 und 6502 zeigt das Bild oben.

Unterprogramme

Die Anweisung „CALL Adresse" beim Z80 oder „JSR Adresse" (von englisch jump to subroutine = springe zum Unterprogramm) beim 6502 ist der Aufruf zu einem Unterprogramm ähnlich wie GOSUB in BASIC. So wie dort ist auch hier eine Return-Anweisung nötig (RET beim Z80 und RTS beim 6502).



Weist man den Computer an, ein Unterprogramm abzuarbeiten, so legt er den augenblicklichen Inhalt des Programmzählers auf den Maschinenstapel, einen für den Computer reservierten Speicherbereich (siehe Seite 10). Danach lädt er die Adresse des Unterprogramms in den Programmzähler.

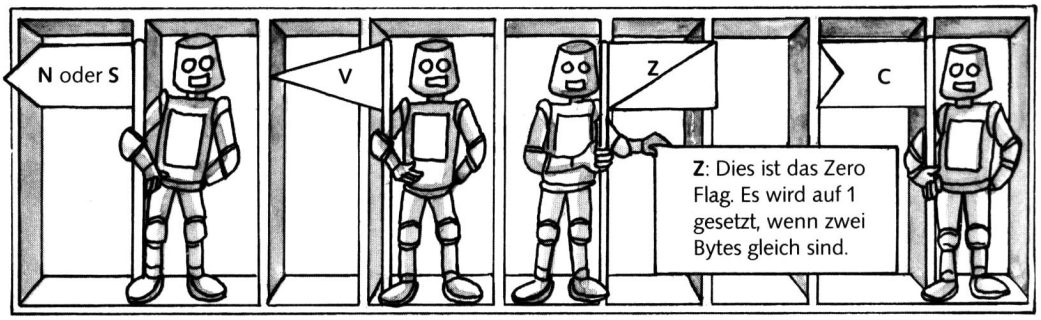


Erreicht der Computer am Ende des Unterprogramms die Anweisung RTS oder RET, dann holt er die Ausgangsadresse wieder vom Stapel in den Programmzähler.

Das ist die Adresse der Anweisung, die auf das Unterprogramm folgt. Auf die gleiche Art werden auch Maschinencode-Programme innerhalb eines BASIC-Programms aufgerufen.

Bedingte Verzweigungen

Bei einer bedingten Verzweigung prüft der Computer eines der Bits im Flagregister. Je nachdem, welches Ergebnis er erhält, verzweigt er oder fährt mit der nächsten Anweisung fort. Im folgenden sind die Bits des Kennzeichenregisters aufgeführt, die man für eine Verzweigung prüfen kann.



N oder **S**: Dies ist das Vorzeichenbit: N beim 6502 und S beim Z80. Es wird 0 bei einem positiven Rechenergebnis und 1 bei einem negativen.

V oder **P/V**: Das ist das Überlaufbit beim 6502. Beim Z80 hat es zwei Bedeutungen und heißt Paritäts-/Überlaufbit. Das Überlaufbit wird 1, wenn eine Rechnung mit Zweierkomplement (siehe nächste Seite) einen Überlauf in das Vorzeichenbit ergibt. Als Paritäts-Prüfbit wird es 1, wenn ein Byte eine ungerade Anzahl von 1en enthält; man braucht es für Kontrollzwecke.

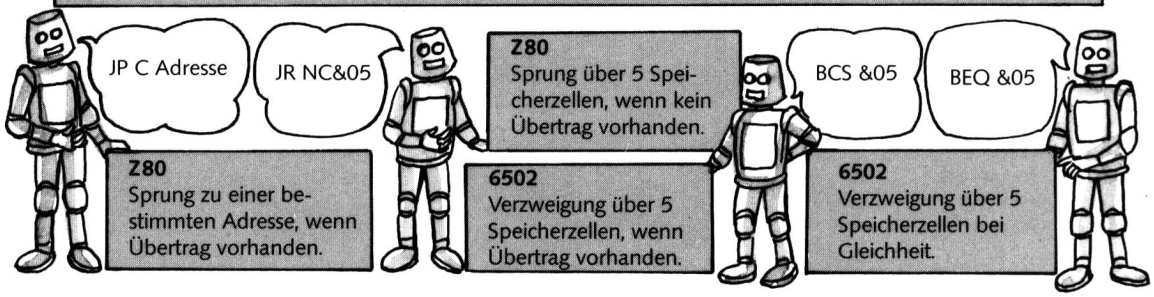
C: Das ist das Übertragskennzeichen oder Carry Flag. Es wird auf 1 gesetzt, wenn das Ergebnis einer Addition nicht in ein Byte paßt.

Außer der Vergleichsanweisung beeinflussen noch andere Anweisungen die Kennzeichen. Die Anweisung DEC (von englisch **dec**rement = verringere) des 6502 beeinflußt z. B. das Vorzeichen und das Zero Flag*.

Opcodes für bedingte Verzweigungen

Hier folgen Verzweigungsanweisungen zur Prüfung der einzelnen Kennzeichen.

Z80 Sprung, wenn ...		6502 Verzweigung, wenn ...	
JP C	Übertrag (C = 1)	BCS	Übertrag (C = 1)
JP NC	kein Übertrag (C = 0)	BCC	kein Übertrag (C = 0)
JP Z	Gleich (Z = 1)	BEQ	Gleich (Z = 1)
JP NZ	Nicht gleich (Z = 0)	BNE	Nicht gleich (Z = 0)
JP M	Minus (S = 1)	BMI	Minus (N = 1)
JP P	Plus (S = 0)	BPL	Plus (N = 0)
JP PO	Parität ungerade (P/V = 1)	BVS	Überlauf (V = 1)
JP PE	Parität gerade (P/V = 0)	BVC	kein Überlauf (V = 0)



Beim Z80 gibt man dem Computer nach der Anweisung „JP, wenn ..." die Adresse an, zu der er springen soll. Beim 6502 gibt man dem Computer die Zahl der Speicherzellen an, die er vorwärts oder rückwärts überspringen soll. Das nennt man *relative Adressierung*, und die Zahl heißt Distanzadresse (englisch displacement oder offset).

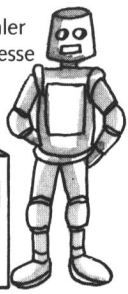
Der Z80 hat außer der Anweisung JP noch die Anweisung „JR, wenn ...", auf die statt einer Adresse eine Distanzadresse folgen kann. Bei dieser Anweisung können aber nur Zero Flag und Übertragskennzeichen geprüft werden.

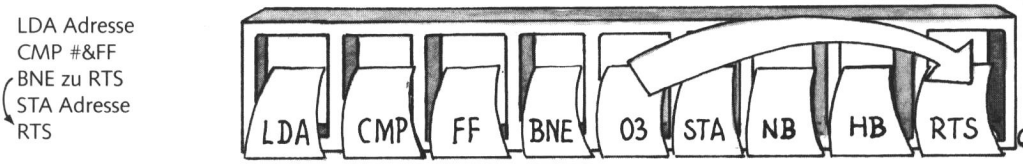
* Eine vollständige Anweisungsliste für Ihren Mikroprozessor gibt auch Auskunft darüber, welche Kennzeichen

Für Adressen müssen zwei Bytes gezählt werden.

So bestimmt man eine Distanzadresse

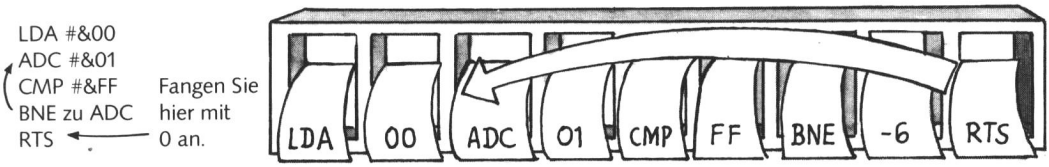
Die Zieladresse einer Verzweigung erhält der Computer bei relativer Adressierung, indem er die Distanzadresse zum Inhalt des Programmzählers addiert oder davon abzieht. Die Distanzadresse erhalten Sie, wenn Sie die Bytes abzählen, die übersprungen werden sollen. Sie beginnen mit 0 bei der Adresse, die der Programmzähler augenblicklich enthält – das ist die Adresse der Anweisung nach der Verzweigung –, und zählen die Zieladresse mit. Unten sind zwei Beispielprogramme für den 6502. Beim Z80 funktioniert das genauso.





Im Beispiel oben verzweigt der Computer mit der Distanzadresse 03 von BNE zu RTS.

Im Beispiel unten verzweigt der Computer mit der Distanzadresse –6 zurück von BNE zu ADC.



Sprünge – vorwärts und rückwärts

Bei Vorwärtssprüngen wandelt man die Distanzadresse in Hex um und setzt sie in das Programm ein. Bei Rückwärtssprüngen ist die Distanzadresse jedoch eine negative Zahl, die man in einem Hex-Byte darstellen muß. Dazu dient das Zweierkomplement. In dieser Darstellungsart gilt das linke Bit als Vorzeichen. Wenn es 1 ist, dann ist die Zahl negativ; ist es 0, dann ist sie positiv.

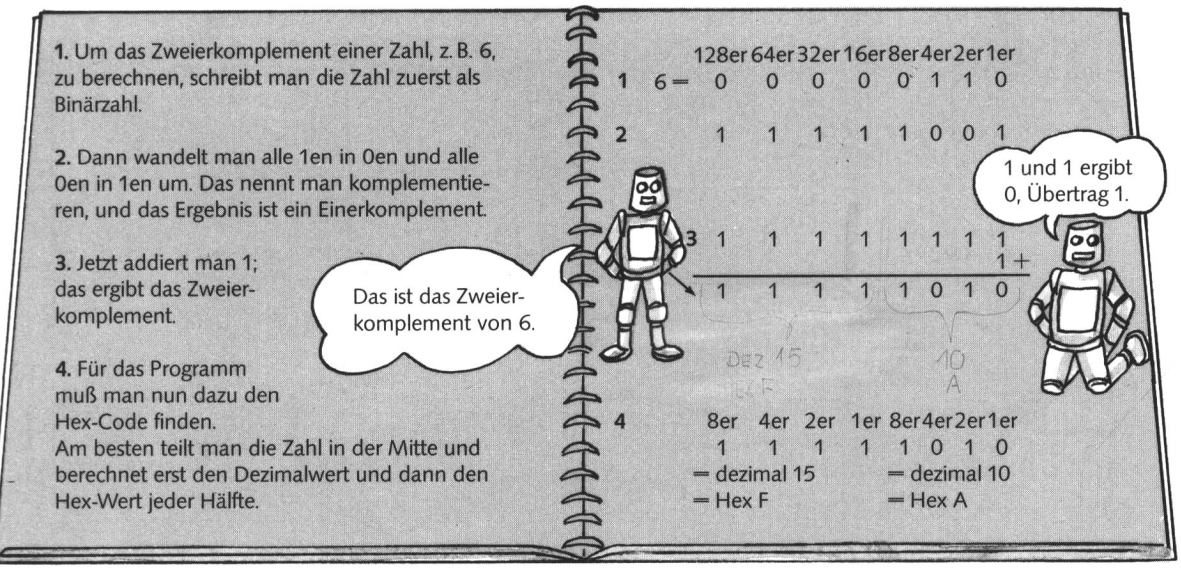
Das Zweierkomplement

1. Um das Zweierkomplement einer Zahl, z. B. 6, zu berechnen, schreibt man die Zahl zuerst als Binärzahl.

2. Dann wandelt man alle 1en in 0en und alle 0en in 1en um. Das nennt man komplementieren, und das Ergebnis ist ein Einerkomplement.

3. Jetzt addiert man 1; das ergibt das Zweierkomplement.

4. Für das Programm muß man nun dazu den Hex-Code finden.
Am besten teilt man die Zahl in der Mitte und berechnet erst den Dezimalwert und dann den Hex-Wert jeder Hälfte.



Das Zweierkomplement von 6 in Hex ist also FA. Bei einem Rückwärtssprung schreibt man das in das Programm. Sobald der Computer hier als erstes Bit 1 findet, weiß er, daß die folgende Zahl negativ ist. Da beim Zweierkomplement ein Bit das Vorzeichen darstellt, bleiben sieben Bits für den Zahlenwert. Damit ist die größte negative Zahl 128 und die größte positive 127.

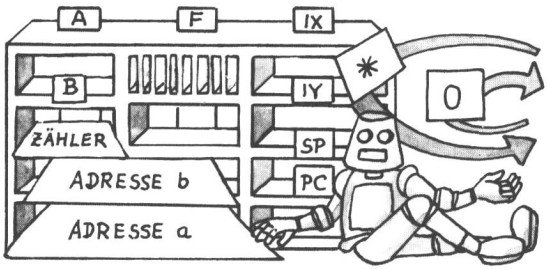
Versuchen Sie, den Hex-Code für das Zweierkomplement von 12, 18 und 9 zu berechnen. (Lösung auf Seite 46.)

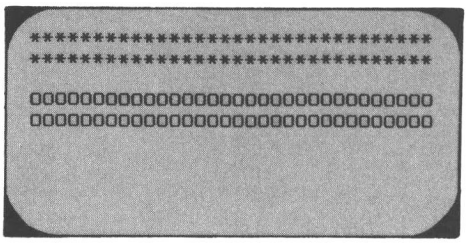
Programm „Bildschirm-Blinken"

Auf diesen beiden Seiten wird ein Programm entwickelt, mit dem man die Bildschirmanzeige blinken lassen kann: Dabei werden zwei Anzeigeblocks gegeneinander ausgetauscht. Das Programm für den Z80 steht auf dieser Seite, das Programm für den 6502 auf der nächsten Seite. Im Anschluß daran finden Sie eine Anleitung, wie Sie das Programm zum Laufen bringen.

Blinken beim Z80

Einfach gesagt, tauscht das Programm die Anzeigeblocks aus, indem es jeden Block Byte für Byte in die Register holt und dann das Byte aus Block a in eine Speicherzelle für Block b ablegt und umgekehrt.



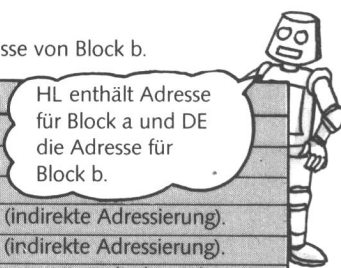


Das Programm verwendet die indirekte Adressierung. Die Bildschirmspeicher-Adressen der ersten Bytes aus jedem Block werden in die Registerpaare HL und DE geladen. Diese Adressen braucht der Computer, um die Bytes holen und speichern zu können. Nach dem Austausch der Bytes erhöht die Anweisung INC den Inhalt von HL und DE um 1, so daß die Register beim nächsten Programmdurchlauf die Adressen der nächsten Bytes enthalten.

Register B enthält die Anzahl der Bytes, die ausgetauscht werden sollen. Bei jedem Durchlauf wird der Inhalt von B um 1 verringert (dekrementiert). Ist B=0, dann sind alle Bytes ausgetauscht.

Das Programm für den Z80

n = Anzahl der Bytes in einem Block; a = erste Adresse von Block a; b = erste Adresse von Block b.



Mnemonics	Hex-Codes	Bedeutung
LD B, n	0,6 n	Zähler
LD HL, (Adresse a)	21, Adresse a	Lädt HL mit Blockadresse a.
LD DE, (Adresse b)	11, Adresse b	Lädt DE mit Blockadresse b.
LD C, (HL)	4E	Lädt C mit Byte von Adresse aus HL (indirekte Adressierung).
LD A, (DE)	1A	Lädt A mit Byte von Adresse aus DE (indirekte Adressierung).
LD (HL), A	77	Speichert Inhalt des Akkus in Adresse aus HL (indirekte Adr.).
LD A, C	79	Lädt Akkumulator mit Byte aus C (erstes Byte von Block a).
LD (DE), A	12	Speichert Inhalt des Akkumulators in Adresse aus DE.
INC HL INC DE	23 13	Addiert 1 zu HL und DE.
DEC B	05	Subtrahiert 1 von B, dem Zähler.
LD A, &00	3E, 00	Lädt Akkumulator mit 0.
CP B	B8	Vergleicht B mit dem Inhalt des Akkumulators (0).
JR NZ zur 4. Anweisung	20, F3	Springt &F3 Speicherzellen zurück, wenn B nicht 0 ist, und lädt neue Bytes in die Register. F3 ist in Hex das Zweierkomplement von 13 (siehe Seite 37).
RET	C9	Return

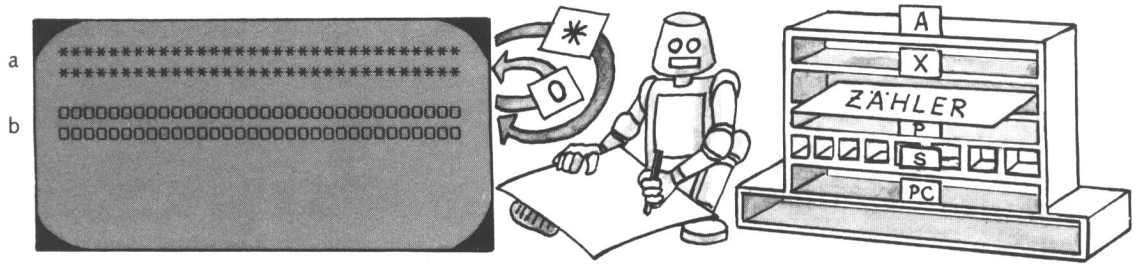
Daten und Adressen einfügen

n (Anzahl von Zeichen in einem Block): Errechnet sich aus der Anzahl der Buchstaben pro Zeile mal Zeilenzahl pro Block. In Hex umwandeln.

Adresse a und b: Will man die beiden obersten Zeilen mit den beiden nächsten vertauschen, dann wird Adresse a zur ersten des Bildschirmspeichers. Für Adresse b muß man zu Adresse a die Zahl der Zeichen in einem Block addieren.

Blinken beim 6502

Das Programm tauscht die beiden Blocks Byte für Byte (d. h. Zeichen für Zeichen); es beginnt beim letzten Byte jedes Blocks. Es holt die Bytes in die Register und speichert sie jeweils unter der Adresse, aus der das andere kommt. Der nächste Programmdurchlauf tauscht die nächsten Bytes usw.



Das Programm verwendet die indizierte Adressierung. Die Anzahl der Bytes in einem Block wird in das Register X geladen. Um Bytes zu holen oder zu speichern, wird die Zahl in X zu den Anfangsadressen der Blocks addiert. Die Anweisung DEX verringert X jeweils um 1, so daß beim nächsten Programmdurchlauf die Bytes der nächstniedrigeren Adresse getauscht werden.

Das Programm für den 6502

Auf der vorhergehenden Seite unten steht, wie man die Werte für n, a und b findet. Von a und b muß man noch 1 subtrahieren, damit der Computer beim Addieren von X die letzte Blockadresse findet und nicht die erste Adresse in der nächsten Zeile nimmt. Natürlich müssen n, a und b in Hex-Ziffern umgewandelt werden.

Mnemonics	Hex-Codes	Bedeutung
LDX #n	A2 n	Lädt X mit Anzahl der Bytes in einen Block.
LDA Adresse a, X	BD Adresse a	Lädt Akkumulator mit Byte in Adresse a+X.
TAY	A8	Überträgt Inhalt des Akkumulators in Register Y.
LDA Adresse b, X	BD Adresse b	Lädt Akkumulator mit Byte in Adresse b+X.
STA Adresse a, X	9D Adresse a	Speichert Inhalt des Akkumulators in Adresse a+X.
TYA	98	Überträgt Inhalt von Register Y in den Akkumulator.
STA Adresse b, X	9D Adresse b	Speichert Inhalt des Akkumulators in Adresse b+X.
DEX	CA	Verringert X. Zero Flag wird 1, wenn X=0.
BNE zur 2. Anweisung	D0 EF	Verzweigt &EF Speicherzellen zurück, falls X nicht gleich 0. EF ist in Hex das Zweierkomplement von 17 (siehe Seite 37).
RTS	60	Return

So läßt man die Programme ablaufen (für Z80 und 6502)

Am besten läßt man das Programm als Maschinencode-Unterprogramm des Hex-Laders ablaufen. Und so können Sie das machen:

1. Geben Sie den Hex-Lader und den Maschinencode für Ihren Mikroprozessor (Zeile 160) ein.

2. Ab Zeile 180 sind zwei Schleifen nötig, um mit POKE die Zeichen im Bildschirmspeicher abzulegen. Das Beispiel unten ergibt je zwei Reihen aus 40 Sternchen (Code 42) und 40 Os (Code 48).

```
180 FOR J=0 TO 79
190 POKE 1.BILDSCHIRM-
SPEICHER-ADRESSE + J,42
200 NEXT J
210 FOR J=80 TO 159
220 POKE 2.BILDSCHIRM-
SPEICHER-ADRESSE + J,48
230 NEXT J
```

3. Nach diesem Programmteil geben Sie die folgenden Zeilen ein:

```
240 CALL ERSTE ADRESSE
    DES MASCHINENCODES
250 FOR K=1 TO 500
260 NEXT K
270 GOTO 240
```

Die Zahl 500 bestimmt die Länge der Verzögerung.

4. Jetzt starten Sie das Programm mit RUN. Der Hex-Lader bringt die Hex-Codes in den Speicher, und die zwei Schleifen bringen die Zeichencodes in den Bildschirmspeicher. Zeile 240 läßt den Computer zur Adresse des Maschinencodes springen und die Anweisungen dort ausführen. Damit sind die Blocks aber nur einmal ausgetauscht. Daher schickt Zeile 270 den Computer immer wieder zurück, um einen Blinkeffekt zu erzeugen. Die Schleife ist zur Verzögerung nötig, weil das Maschinencode-Programm so schnell ist.

Wie man weiterkommt

Mehr über Maschinencode lernen Sie am ehesten, wenn Sie eigene kleine Programme schreiben oder sich mit fremden Programmen beschäftigen. Maschinencode eignet sich gut für Unterprogramme, mit denen bestimmte Aufgaben in BASIC-Programmen erledigt werden sollen. Das ist besonders sinnvoll, wenn man Daten sortieren oder den Bildschirm mit Grafik füllen möchte. Maschinencode läuft nämlich schneller als BASIC und braucht weniger Speicherplatz. Unterprogramme für diese Zwecke findet man häufig in Zeitschriften. Sind sie speziell für Ihren Computer geschrieben, so kann man sie ohne Änderung übernehmen. Sind sie für einen anderen Computer mit dem gleichen Mikroprozessor geschrieben, dann müssen Sie alle Adressen für Ihren Computer anpassen, beispielsweise die Adresse des Bildschirmspeichers oder die Adressen, unter denen der Maschinencode gespeichert wird.

Maschinencode-Unterprogramme

So können Sie Maschinencode-Unterprogramme in BASIC-Programmen verwenden:

1. Reservieren Sie Speicherplatz für den Maschinencode (siehe Seite 20 bis 22).

2. Geben Sie den Maschinencode in Zeile 160 des Hex-Lader-Programms (Seite 24) ein. Vergessen Sie nicht die Anweisung RET oder RTS am Ende des Maschinencodes! Falls noch Daten gespeichert werden müssen (mit POKE), dann fügen Sie die entsprechenden Zeilen hinzu. Starten Sie den Hex-Lader.

3. Lassen Sie Ihr BASIC-Programm mit einer Zeilennummer nach den Zeilennummern des Hex-Laders beginnen. Dort wo der Computer das Maschinencode-Programm ausführen soll, schreiben Sie die entsprechende Anweisung für Ihren Computer in BASIC hin.

4. Starten Sie Ihr BASIC-Programm. Der Computer wird die BASIC-Anweisungen ausführen, und wenn er die Zeile erreicht, in der das Maschinencode-Unterprogramm aufgerufen wird, springt er zur dort angegebenen Adresse und führt die Maschinencode-Anweisungen aus. Die Anweisung „Return" schickt ihn zurück zur nächsten Zeile des BASIC-Programms.

Hilfe durch Assembler

Ein Assembler, also ein Programm, mit dem man Maschinencode-Programme in Mnemonics eingeben kann, erleichtert das Programmieren in Maschinencode erheblich. Für die meisten Heimcomputer gibt es Assembler auf Kassetten, bei manchen ist er bereits eingebaut.

Mit dem Assembler kann man zu den Zeilen Kommentare eingeben, die einem sagen, was dort geschieht. Er zeigt dann auf dem Bildschirm das Programm in Hex-Codes und Mnemonics, zusammen mit den Speicheradressen und Kommentaren.

Der Assembler vertauscht automatisch die Ziffernpaare von Adressen und berechnet bei Sprüngen das Ziel oder die Distanzadresse. Bei einigen Assemblern kann man den Daten symbolische Namen geben, ähnlich wie den Variablen in BASIC. Ein guter Assembler leistet Hilfestellung bei Fehlersuche und Korrektur.

Weitere Informationen

In Computer-Zeitschriften finden Sie sicher immer wieder einmal Besprechungen von Büchern über bestimmte Mikroprozessoren. Nützlich dürften Ihnen vor allem die folgenden Titel sein:

R. Zaks: *Programmierung des Z80.* Düsseldorf 1982, Sybex-Verlag,

R. Zaks: *Programmierung des 6502.* Düsseldorf 1982, Sybex-Verlag.

Diese beiden Bücher enthalten detaillierte Anleitungen mit vollständigen Listen des jeweiligen Befehlsvorrats; sie sind für den Anfänger nicht leicht zu lesen, aber sehr praktisch zum Nachschlagen.

Umwandlungstabellen für Dezimal- und Hex-Zahlen

Mit Hilfe der ersten Tabelle wandelt man Hex-Zahlen von 00 bis FF in Dezimalzahlen um und umgekehrt.

Hex in Dezimal
Zur Umwandlung von Hex-Zahlen suchen Sie in der Zeile für die erste Hex-Ziffer die Spalte für die zweite. Am Schnittpunkt von Zeile und Spalte steht der gesuchte Dezimalwert. Z. B. ist Hex A1 dezimal 161.

Dezimal in Hex
Zur Umwandlung einer Dezimalzahl in eine Hex-Zahl suchen Sie in der Tabelle die Dezimalzahl. Am linken Rand der Zeile steht die erste Hex-Ziffer, über der Spalte die zweite. Z. B. ist dezimal 154 in Hex 9A.

Erste Hex-Ziffer \ Zweite Hex-Ziffer	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Umwandlung von Adressen
Suchen Sie zuerst den Dezimalwert des ersten Ziffernpaars der Hex-Adresse. Das ist die Seitenzahl (siehe Seite 11). Der Dezimalwert des zweiten Ziffernpaars gibt die Stelle auf der Seite an. Multiplizieren Sie die Seitenzahl mit 256 und addieren Sie dann den Wert der Stelle auf der Seite.

Zweierkomplement-Tabelle

Mit Hilfe der folgenden Tabelle finden Sie die Hex-Ziffern des Zweierkomplements zu den Dezimalzahlen –1 bis –128. Suchen Sie die Dezimalzahl in der Tabelle. Die „Zeilennummer" ergibt die erste Hex-Ziffer, die „Spaltennummer" die zweite.

Erste Hex-Ziffer \ Zweite Hex-Ziffer	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
F	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
E	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
D	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
C	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
B	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
A	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
9	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
8	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128

Umwandlung mit dem Taschenrechner

Angenommen, Sie wollen die Dezimalzahl 134 in Hex-Ziffern umwandeln. Dazu teilen Sie 134 durch 16; der Rechner zeigt 8,375 an. Die Zahl vor dem Komma ist die Anzahl der 16er und liefert die erste Hex-Ziffer.

Um den Rest als ganze Zahl zu erhalten, multiplizieren Sie die Ziffern hinter dem Komma mit 16.

Beispiel: 134 : 16 = 8,375; 0,375 · 16 = 6

Also ist 134 : 16 = 8 Rest 6, und dezimal 134 ist in Hex 86.

Mnemonics und Hex-Codes für den Z80

Auf dieser und den beiden nächsten Seiten finden Sie die Mnemonics und Hex-Codes für die Anweisungen, die in diesem Buch behandelt wurden. Der Ausdruck *implizite Adressierung*, der hier benutzt wird, bedeutet, daß schon der Opcode den Operanden bezeichnet. Wenn Sie weiter mit Maschinencode arbeiten wollen, brauchen Sie eine vollständige Liste des Befehlsvorrats (siehe „Weitere Informationen" auf Seite 40), denn hier können nicht alle Anweisungen für den Z80 aufgeführt werden. Die Abkürzungen, die Sie in den anschließenden Tabellen finden, haben folgende Bedeutungen:

n = Zahl
nn = Zwei-Byte-Zahl
r = Register
rr = Registerpaar
x = Adresse
c = Bedingung
d = Distanzadresse

ADC A, n: Addiert n und Übertrag zum Akkumulator.
(Unmittelbare Adressierung)

ADC A, n	CE, n

ADC A, r: Addiert n und Übertrag zum Akkumulator.
(Implizite Adressierung)

ADC A, A	8F
ADC A, B	88
ADC A, C	89
ADC A, D	8A
ADC A, E	8B
ADC A, H	8C
ADC A, L	8D

ADC HL, rr: Addiert Registerpaar rr und Übertrag zu HL.
(Implizite Adressierung)

ADC HL, BC	ED 4A
ADC HL, DE	ED 5A
ADC HL, HL	ED 6A

ADD A, n: Addiert n zum Akkumulator.
(Unmittelbare Adressierung)

ADD, n	C6, n

ADD A, r: Addiert Register r zum Akkumulator.
(Implizite Adressierung)

ADD A, A	87
ADD A, B	80
ADD A, C	81
ADD A, D	82
ADD A, E	83
ADD A, H	84
ADD A, L	85

ADD HL, rr: Addiert Registerpaar rr zu HL.
(Implizite Adressierung)

ADD HL, BC	09
ADD HL, DE	19
ADD HL, HL	29

CALL x: Aufruf des Unterprogramms mit Adresse x.
(Unmittelbare Adressierung)

CALL x	CD x

CALL c, x: Aufruf des Unterprogramms mit Adresse x, falls Bedingung c zutrifft. c kann sein: Z (gleich), NZ (nicht gleich), C (Übertrag), NC (kein Übertrag), PE (gerade Parität), PO (ungerade Parität), M (minus), P (plus).
(Unmittelbare Adressierung)

CALL Z, x	CC, x
CALL NZ, x	C4, x
CALL C, x	DC, x
CALL NC, x	D4, x
CALL PE, x	EC, x
CALL PO, x	E4, x
CALL M, x	FC, x
CALL P, x	F4, x

CCF: Komplementiert Übertragskennzeichen.
(Implizite Adressierung)

CCF	3 F

CP n: Vergleicht den Inhalt des Akkumulators mit Datum n.
(Unmittelbare Adressierung)

CP n	FE n

CP r: Vergleicht Register r mit dem Akkumulator.
(Implizite Adressierung)

CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD

CP (HL): Vergleicht Inhalt der Adresse in HL mit dem Akkumulator.
(Indirekte Adressierung)

CP (HL)	BE

DEC r: Subtrahiert 1 von Register r.
(Implizite Adressierung)

DEC A	3D
DEC B	05
DEC C	0D
DEC D	15
DEC E	1D
DEC H	25
DEC L	2D

DEC rr: Subtrahiert 1 von Registerpaar rr.
(Implizite Adressierung)

DEC BC	0B
DEC DE	1B
DEC HL	2B
DEC IX	DD2B
DEC IY	FD2B

DEC (HL): Subtrahiert 1 vom Inhalt der Adresse in HL.
(Indirekte Adressierung)

DEC (HL)	35

INC r: Register r plus 1.
(Implizite Adressierung)

INC A	3C
INC B	04
INC C	0C
INC D	14
INC E	1C
INC H	24
INC L	2C

INC rr: Registerpaar rr plus 1.
(Implizite Adressierung)

INC BC	03
INC DE	13
INC HL	23

INC (HL): Inhalt von Adresse in HL plus 1.
(Indirekte Adressierung)

INC (HL)	34

JP x: Sprung zu Adresse x.
(Unmittelbare Adressierung)

JP x	C3 x

JP (rr): Sprung zur Adresse in Registerpaar rr. (Implizite Adressierung)

JP (HL)	E9
JP (IX)	DD E9
JP (IY)	FD E9

JP c, x: Sprung zu Adresse x, falls Bedingung c zutrifft. c kann sein: Z (gleich), NZ (nicht gleich), C (Übertrag), NC (kein Übertrag), PE (gerade Parität), PO (ungerade Parität), M (minus), P (plus). (Unmittelbare Adressierung)

JP Z, x	CA, x
JP NZ, x	C2, x
JP C, x	DA, x
JP NC, x	D2, x
JP PE, x	EA, x
JP PO, x	E2, x
JP M, x	FA, x
JP P, x	F2, x

JR d: Sprung über d Speicherzellen. (Relative Adressierung)

JR d	18 d

JR c, d: Sprung über d Speicherzellen, falls Bedingung c zutrifft. c kann sein: Z (gleich), NZ (nicht gleich),C (Übertrag), NC (kein Übertrag). (Relative Adressierung)

JR NZ, d	20, d
JR Z, d	28, d
JR NC, d	30, d
JR C, d	38, d

LD r, n: Lädt Zahl n in Register r. (Unmittelbare Adressierung)

LD A, n	3E, n
LD B, n	06, n
LD C, n	0E, n
LD D, n	16, n
LD E, n	1E, n
LD H, n	26, n
LD L, n	2E, n

LD rr, nn: Lädt Zahl nn in Registerpaar rr. (Unmittelbare Adressierung)

LD BC, nn	01, nn
LD DE, nn	11, nn
LD HL, nn	21, nn

LD A, (x): Lädt Inhalt von Adresse x in den Akkumulator. (Direkte Adressierung)

LD A, (x)	3A, (x)

LD rr, (x): Lädt Inhalt von Adresse x und x+1 in Registerpaar rr. (Direkte Adressierung)

LD BC, (x)	ED 4B, (x)
LD DE, (x)	ED 5B, (x)
LD HL, (x)	2A, (x)

LD A, r: Lädt Inhalt von Register r in den Akkumulator. (Implizite Adressierung)

LD A, A	7F
LD A, B	78
LD A, C	79
LD A, D	7A
LD A, E	7B
LD A, H	7C
LD A, L	7D

LD B, r: Lädt Inhalt von Register r in Register B. (Implizite Adressierung)

LD B, A	47
LD B, B	40
LD B, C	41
LD B, D	42
LD B, E	43
LD B, H	44
LD B, L	45

LD C, r: Lädt Inhalt von Register r in Register C. (Implizite Adressierung)

LD C, A	4F
LD C, B	48
LD C, C	49
LD C, D	4A
LD C, E	4B
LD C, H	4C
LD C, L	4D

LD D, r: Lädt Inhalt von Register r in Register D. (Implizite Adressierung)

LD D, A	57
LD D, B	50
LD D, C	51
LD D, D	52
LD D, E	53
LD D, H	54
LD D, L	55

LD E, r: Lädt Inhalt von Register r in Register E. (Implizite Adressierung)

LD E, A	5F
LD E, B	58
LD E, C	59
LD E, D	5A
LD E, E	5B
LD E, H	5C
LD E, L	5D

LD H, r: Lädt Inhalt von Register r in Register H. (Implizite Adressierung)

LD H, A	67
LD H, B	60
LD H, C	61
LD H, D	62
LD H, E	63
LD H, H	64
LD H, L	65

LD L, r: Lädt Inhalt von Register r in Register L. (Implizite Adressierung)

LD L, A	6F
LD L, B	68
LD L, C	69
LD L, D	6A
LD L, E	6B
LD L, H	6C
LD L, L	6D

LD r, (rr): Lädt Inhalt von Adresse in Registerpaar rr in Register r. (Indirekte Adressierung)

LD A, (BC)	0A
LD A, (DE)	1A
LD A, (HL)	7E
LD B, (HL)	46
LD C, (HL)	4E
LD D, (HL)	56
LD E, (HL)	5E
LD H, (HL)	66
LD L, (HL)	6E

LD (x), A: Speichert Inhalt des Akkumulators unter Adresse x. (Direkte Adressierung)

LD (x), A	32, x

LD (x), rr: Speichert Inhalt von Registerpaar rr unter Adresse x und x+1. (Direkte Adressierung)

LD (x), BC	ED 43, x
LD (x), DE	ED 53, x
LD (x), HL	22, x

LD (rr), r: Speichert Inhalt von Register r unter Adresse in Registerpaar rr.
(Indirekte Adressierung)

LD (BC), A	02
LD (DE), A	12
LD (HL), A	77
LD (HL), B	70
LD (HL), C	71
LD (HL), D	72
LD (HL), E	73
LD (HL), H	74
LD (HL), L	75

LD (rr), n: Speichert Zahl n unter Adresse in Registerpaar rr.
(Unmittelbare/indirekte Adressierung)

LD (HL), n	36

RET: Zurück vom Unterprogramm.
(Indirekte Adressierung)

RET	C9

RET c: Zurück vom Unterprogramm, falls Bedingung c zutrifft. c kann sein: Z (gleich), NZ (nicht gleich), C (Übertrag), NC (kein Übertrag), PE (gerade Parität), PO (ungerade Parität), P (plus), M (minus).
(Indirekte Adressierung)

RET Z	C8
RET NZ	C0
RET C	D8
RET NC	D0
RET PE	E8
RET PO	E0
RET M	F8
RET P	F0

SBC A, n: Subtrahiert Zahl n und Übertrag vom Akkumulator.
(Unmittelbare Adressierung)

SBC A, n	DE, n

SBC A, r: Subtrahiert Inhalt von Register r und Übertrag vom Akkumulator.
(Implizite Adressierung)

SBC A, A	9F
SBC A, B	98
SBC A, C	99
SBC A, D	9A
SBC A, E	9B
SBC A, H	9C
SBC A, L	9D

SBC HL, rr: Subtrahiert Inhalt von Registerpaar rr und Übertrag von HL.
(Implizite Adressierung)

SBC HL, BC	ED 42
SBC HL, DE	ED 52
SBC HL, HL	ED 62

SBC A, (HL): Subtrahiert Inhalt von Adresse in HL und Übertrag vom Akkumulator.
(Indirekte Adressierung)

SBC A, (HL)	9E

SCF: Setzt Übertragskennzeichen auf 1.
(Implizite Adressierung)

SCF	37

SUB n: Subtrahiert Zahl n vom Akkumulator.
(Unmittelbare Adressierung)

SUB, n	D6, n

SUB r: Subtrahiert Inhalt von Register r vom Akkumulator.
(Implizite Adressierung)

SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB L	95

SUB (HL): Subtrahiert Inhalt der Adresse HL vom Akkumulator.
(Indirekte Adressierung)

SUB (HL)	96

Änderungen für den Hex-Lader (Seite 24)

Ändern Sie diese Zeilen für den Sinclair ZX81 (Timex 1000):

```
40 INPUT H$
70 LET X=(CODE(H$)-28)*16
80 ---ENTFAELLT---
90 LET Y=CODE(H$(2 TO ))-28
100 LET X=X+Y
110 ---ENTFAELLT---
155 ---ENTFAELLT---
160 ---ENTFAELLT---
```

Ändern Sie diese Zeilen für Atari-Computer:

```
90 LET Y=ASC(A$(2))
```

Mnemonics und Hex-Codes für den 6502

Diese Tabelle zeigt die Mnemonics und Hex-Codes aller Anweisungen für den 6502, die in diesem Buch vorkommen (und einige mehr). Links stehen die Mnemonics und in der rechten Hälfte die Hex-Codes für die verschiedenen Adressierungsarten. Die *Null-Seiten-Adressierung* ist eine Art von direkter Adressierung, wobei das höherwertige Adreßbyte immer 0 ist, die angesprochene Adresse also immer auf der ersten Speicherseite liegt (siehe Seite 10). *Implizite Adressierung* bedeutet, daß schon der Opcode einer Anweisung den Operanden bezeichnet, z. B. CLC. Wenn Sie weiter mit Maschinencode arbeiten wollen, sollten Sie sich eine vollständige Liste des Befehlsvorrats besorgen (siehe „Weitere Informationen" auf Seite 40), da hier nicht alle Anweisungen für den 6502 aufgeführt werden können.

Adressierungsarten	Unmittelbar	Direkt	Null-Seite	Indiziert X	Indiziert Y	Implizit	Relativ
Operand ist	Datum	Eine Adresse	Adresse auf Null-Seite	Adresse + Register X	Adresse + Register Y	Keiner	Distanzadresse
ADC: Addiert ein Byte und den Übertrag zum Akkumulator.	69	6D	65	7D	79		
BCC: Verzweigt, wenn Übertrag 0.							90
BCS: Verzweigt, wenn Übertrag 1.							B0
BEQ: Verzweigt, wenn gleich.							F0
BMI: Verzweigt, wenn minus.							30
BNE: Verzweigt, wenn nicht gleich.							D0
BPL: Verzweigt, wenn plus.							10
BVC: Verzweigt, wenn Überlauf 0.							50
BVS: Verzweigt, wenn Überlauf 1.							70
CLC: Setzt Übertragskennzeichen auf 0.						18	
CMP: Vergleicht mit Akkumulator.	C9	CD	C5	DD	D9		
CPX: Vergleicht mit Register X.	E0	EC	E4				
CPY: Vergleicht mit Register Y.	C0	CC	C4				
DEC: Subtrahiert 1 von Speicherzelle (dekrementieren).		CE	C6	DE			
DEX: Subtrahiert 1 von Register X.						CA	
DEY: Subtrahiert 1 von Register Y.						88	
INC: Addiert 1 zu Speicherzelle.		EE	E6	FE			
INX: Addiert 1 zu Register X.						E8	
INY: Addiert 1 zu Register Y.						C8	
JMP: Springt zur angegebenen Adresse.		4C					
JSR: Springt zum Unterprogramm an angegebener Adresse.		20					
LDA: Lädt Akkumulator mit Byte.	A9	AD	A5	BD	B9		
LDX: Lädt Register X mit Byte.	A2	AE	A6		BE		
LDY: Lädt Register Y mit Byte.	A0	AC	A4	BC			
RTS: Zurück vom Unterprogramm.						60	
SBC: Zieht Byte und Übertrag vom Akkumulator ab.	E9	ED	E5	FD	F9		
SEC: Setzt Übertragskennzeichen auf 1.						38	
STA: Speichert Inhalt des Akkumulators unter einer Adresse.		8D	85	9D	99		
STX: Speichert Register X unter einer Adresse.		8E	86				
STY: Speichert Register Y unter einer Adresse.		8C	84				
TAX: Überträgt Inhalt des Akkumulators in Register X.						AA	
TAY: Überträgt Inhalt des Akkumulators in Register Y.						A8	
TXA: Überträgt Register X in den Akkumulator.						8A	
TYA: Überträgt Register Y in den Akkumulator.						98	

Beachten Sie, daß nicht jede Anweisung in jeder Adressierungsart benutzt werden kann.

Lösungen

Seite 11
&A7 ist dezimal 167. 513 ist &201 in Hex.

Tip: Das Zweierkomplement einer Zahl erhält man leicht, wenn man die Zahl von 256 subtrahiert und das Ergebnis in Hex umwandelt. (Z. B.: 256 – 6 = 250 = &FA.)

Seite 26
1. 25 + 73 (25 ist &19, und 73 ist &49).

Z80		6502		Bedeutung
Mnemonics	**Hex-Codes**	**Mnemonics**	**Hex-Codes**	
LD A, &19	3E, 19	LDA #&19	A9 19	Lädt &19 in den Akkumulator.
ADD, A, &49	C6, 49	ADC #&49	69 49	Addiert &49 zum Akkumulator.
LD (Adresse), A	32, Adresse	STA Adresse	8D Adresse	Speichert Inhalt des Akkumulators unter einer Adresse.
RET	C9	RTS	60	Return

2. 64 + 12 + 14 (64 ist &40, 12 ist &0C, und 14 ist &0E).

Z80		6502		Bedeutung
Mnemonics	**Hex-Codes**	**Mnemonics**	**Hex-Codes**	
LD A, &40	3E, 40	LDA #&40	A9 40	Lädt &40 in den Akkumulator.
ADD A, &0C	C6, 0C	ADC #&0C	69 0C	Addiert &0C zum Akkumulator.
ADD A, &0E	C6, 0E	ADC #&0E	69 0E	Addiert &0E zum Akkumulator.
LD (Adresse), A	32, Adresse	STA Adresse	8D Adresse	Speichert Inhalt des Akkumulators unter einer Adresse.
RET	C9	RTS	60	Return

Seite 28
00011010 ist dezimal 26.
11111011 ist dezimal 251.
10101010 ist dezimal 170.

	Dezimal		Hex	
	höherwertig	niederwertig	höherwertig	niederwertig
307	1	51	&01	&33
21214	82	222	&82	&DE
759	2	247	&02	&F7
1023	3	255	&03	&FF

Seite 31
Damit man das Programm für Ergebnisse verwenden kann, die größer sind als 255, läßt man die Return-Anweisung weg und fügt die unten aufgeführten Zeilen an. Das Ergebnis kann man mit folgender Anweisung auf den Bildschirm bringen:
PRINT PEEK (Adresse 3) + PEEK (Adresse 4) *256

Z80		6502		Bedeutung
Mnemonics	**Hex-Codes**	**Mnemonics**	**Hex-Codes**	
LD A, &00	3E, 00	LDA #&00	A9 00	Lädt 0 in den Akkumulator.
ADC A, &00	CE, 00	ADC #&00	69 00	Addiert 0 und Übertrag zum Akkumulator.
LD (Adresse 4), A	32, Adresse 4	STA Adresse 4	8D Adresse 4	Speichert den Inhalt des Akkumulators unter Adresse 4.
RET	C9	RTS	60	Return

Seite 37
Das Zweierkomplement für 12 ist in Hex &F4; für 18 ist es &EE, und für 9 ist es &F7.

Register

ADC 29
ADD 29
Adreßbus 6
Adresse 8 – 15, 18, 19, 21 – 23, 25 – 27, 35 – 37, 40
Adressierung(sart) 18, 27, 33, 45
– direkte (absolute) 27
– implizite 42, 45
– indirekte 33, 38
– indizierte 34, 39
– Null-Seiten 45
– relative 36, 37
– unmittelbare 18, 27, 33
Akkumulator 14, 15, 17, 18, 27, 32 – 34
Arithmetisch-logische Einheit (ALU) 14, 15
ASC 24
ASCII-Code 13, 24
Assembler(sprache) 5, 16 – 19, 26, 40
Atari-Computer 3, 24

BAD DATA 24
BASIC-(Programm) 3 – 6, 8, 10, 12, 20, 23, 33, 35, 40
Befehlsvorrat 40, 42, 45
Benutzer-RAM 8 – 10, 20 – 22
Betriebssystem 8 – 10, 12, 13, 20
Bildschirm 9, 12 – 14, 24, 32
Bildschirmspeicher 8, 9, 13, 32 – 34
Binärcode, -system, -zahl 4, 5, 7, 9, 12, 13, 19, 28, 29, 37
Bit 4, 9, 14, 15, 29, 36
Bus 6
Byte 4, 6, 8, 9, 12 – 16, 19 – 23, 26 – 30, 32, 34 – 37
– höherwertiges 19, 21, 22, 28, 30
– niederwertiges 19, 21, 22, 28, 30

CALL 25
Carry Flag 14, 28, 29, 36
Chip 3, 6, 7
CLC 29
CLEAR 21
CODE 24
Commodore 64 3, 7
CPU siehe Zentraleinheit
Cursor 9, 10

Datenbus 6
Datenbyte 23, 26, 27
Dezimalsystem, -wert, zahl 8, 11 – 13, 19, 23 – 26, 28, 37, 41
Distanzadresse 36, 37, 40
Dragon 7
Drucker 10

Einerkomplement 37
END 24 – 26

Fehler 3
Flagregister 14, 15, 29, 36
FOR . . . NEXT 12

GOSUB 10
Grafik 40
Grafikzeichen 10, 22

Hex(adezimal)code, -system, -zahl 5, 8, 11, 12, 16 – 20, 23, 24, 26 – 28, 37, 40 – 42, 45
Hex-Lader 5, 22 – 26, 40
HIMEM 8, 20, 21

Impuls 4, 6, 7, 9
Indexregister 14, 15
Integrierte Schaltung (IC) 7
Interpreter 4, 6, 8, 20

Kennzeichenregister 14
Kilobyte 9, 10

LET 12
LIFO 10

Mnemonic 5, 16 – 19, 26, 40, 42, 45
Monitor 8

Null-Seite 10, 45

Objektcode 18
Operand 16 – 18, 27, 33, 42, 45
Op(erations)code 16 – 19, 32, 36, 42, 45
Oric 3, 7, 21

Paritäts-/Überlaufbit 36
PEEK 12, 22
Platine 6, 7
POKE 12, 13, 23, 25 – 27, 30, 32 – 34, 40
PRINT PEEK 12, 21, 22, 25 – 27
PRINT USR 25
Program Counter 14
Programmzähler 14, 15, 35, 37
Prozessor-Statusregister 15, 29
Puffer 10, 22

Quellcode 18

RAM(-Chip) 6 – 10, 12, 15, 20 – 23, 26
RAMTOP 8, 21
Register 14 – 16, 18, 27, 28, 30, 35
REM 22
RET 23
RETURN/Return 10, 23, 26, 35, 40
ROM(-Chip) 6 – 9, 12, 13, 15
RTS 23
RUN 25

Seite 10, 15, 19, 21 – 23, 41
Signal 14
Sinclair-(Timex-)Computer 3, 7, 9, 10, 13, 21, 24, 32
Speicher 6, 8 – 14, 17, 19, 27, 30, 32, 35
Speicheradresse 6, 9, 12, 40
Speicherauszug 19
Speicherplan 8, 12
Speicherplatz 3, 10, 12, 20, 40
Speicherzelle 8 – 13, 20 – 23, 28, 30, 33 – 35
Spiel 3
Sprung 35, 37, 40
Stack Pointer 14
Stapelspeicher 10, 14
Stapelzeiger 14, 15
Steuereinheit 14, 15
SYS 25
Systemvariable 10, 21, 22

Taktgeber 6, 7
Tastatur 6, 8, 10, 13
Textbyte 32 – 34
TI 99/4 7

Überlaufbit 36
Übertragskennzeichen 14, 17, 28, 29, 31, 36
Universalregister 14, 15
Unterprogramm 35, 40

Variable 8, 9, 12
Variablenspeicher 8, 10
VC-20 3, 7, 21, 22
Verzweigung 35 – 37
Vorzeichenbit 36

Zeichencode 32
Zeichenvorrat 13
Zentraleinheit (CPU) 3, 7, 10, 14 – 16, 19
Zero Flag 32 – 34, 36
Zweierkomplement 36, 37, 41

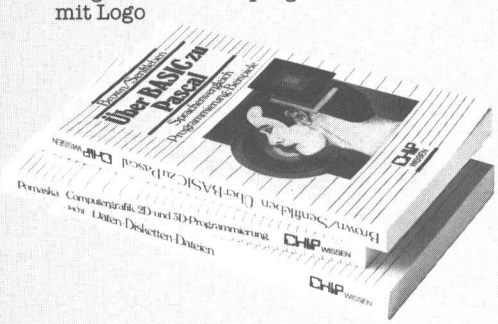
CHIP

Das Mikrocomputer-Magazin mit dem KOMPLETT-PROGRAM informiert umfassend über die faszinierende Welt der Mikrocomputer!

Jeden Monat neu:
Für Einsteiger, Fortgeschrittene und Profis: Spannende Titelstories, Anwendungsbeispiele aus der Praxis, Problemlösungen, Software, Hintergrundberichte. Nachrichten und Trends. Tips und Tricks. Und – die CHIP-Börse, der Kleinanzeigenmarkt.

CHIP-WISSEN, die Buchreihe über Mikrocomputer-Technik:

- Über BASIC zu Pascal
- Computergrafik 2D- und 3D-Programmierung
- Programmieren mit Logo
- Daten · Disketten Dateien
- Programmentwicklung in UCSD-Pascal
- Wie man in BASIC programmiert



CHIP-SOFT- für den Commodore 64, Programme auf Diskette und Cassette

- Schreibmaschinenkurs
- Morsetrainer
- Supervoc

CHIP-Specials – u. a. mit den Siegerprogrammen aus dem Wettbewerb »Goldene Diskette '84«

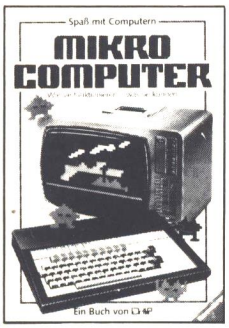
Commodore 64 Programme 3
Sinclair ZX 81 Programme 2
Atari 600XL/800 XL Programme
Sinclair ZX-Spectrum Programme 2
Computerspaß mit C 64
IBM-PC, kompatible und IBM-PCjr. Programme

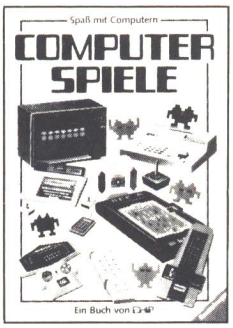
CHIP **Schreiben Sie an den CHIP-Leserservice. Postfach 6740, D-8700 Würzburg, wenn Sie mehr über das komplette Infosystem erfahren möchten.**

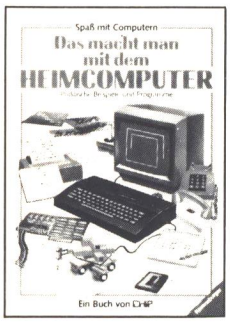
Spaß mit Computern

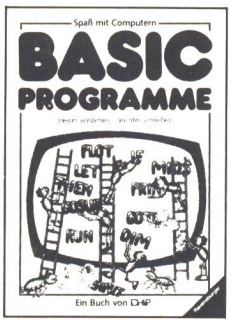
Computer sind kein Hexenwerk. Sie gehören inzwischen zum Alltag in Betrieben und Schulen und für viele auch zur Freizeit. Sie müssen nicht Experte sein oder sich erst mühsam spezielle Fachkenntnisse erwerben, wenn Sie diesem elektronischen Hobby nachgehen wollen. Die reich illustrierten Bände dieser Reihe geben Ihnen in leichtverständlicher Sprache einen Einblick in Technik und Funktionsweisen und zeigen Ihnen die vielfältigen Anwendungsmöglichkeiten von Heimcomputern. Damit können Sie Ihr Interesse an moderner Technik aktiv nutzen – und Sie werden sehen, daß das auch noch Spaß macht!

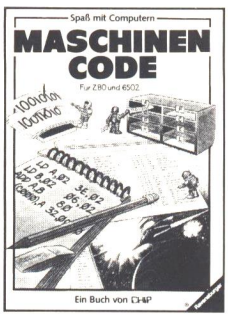
In der Reihe „Spaß mit Computern" sind bisher folgende Titel erschienen:

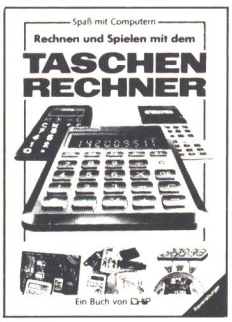


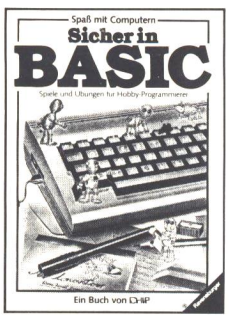


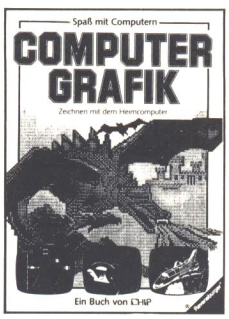












ISBN 3-473-**35613**-1 (Otto Maier Verlag)

ISBN 3-8023-**0822**-0 (Vogel-Verlag)